Algorithmes numériques pour les matrices polynomiales avec applications en commande

(Numerical algorithms for polynomial matrices with applications in control)

Thèse préparée au Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS
en vue de l'obtention du titre de Docteur de
l'Institut National des Sciences Appliquées de Toulouse
par **Juan Carlos Zúñiga Anaya**

Date de la soutenance: le 14 septembre 2005.
Composition du jury:

Germain Garcia, INSA Toulouse (Président)
Didier Henrion, LAAS-CNRS Toulouse (Directeur de these)
Paul Van Dooren, UCL Louvain-la-Neuve (Rapporteur)
Hisham Abou-Kandil, ENS Cachan (Rapporteur)
Michael Šebek, ČVUT Prague (Examinateur)
Javier Ruiz León, CINVESTAV Guadalajara (Examinateur)

Résumé

Dans cette thèse nous développons de nouveaux algorithmes de calcul numérique pour les matrices polynomiales. Nous abordons le problème du calcul de la structure propre (rang, espace nul, structures finie et infinie) d'une matrice polynomiale et nous appliquons les résultats obtenus au calcul de la factorisation J-spectrale des matrices polynomiales. Nous présentons également quelques applications de ces algorithmes en théorie de la commande. Tous les nouveaux algorithmes décrits ici sont basés sur le calcul d'espaces nuls constants de matrices bloc Toeplitz associées à la matrice polynomiale analysée. Pour calculer ces espaces nuls nous utilisons des méthodes standard de l'algèbre linéaire numérique comme la décomposition en valeurs singulières ou la factorisation QR. Nous étudions aussi l'application de méthodes rapides comme la méthode généralisée de Schur pour les matrices structurées. Nous analysons les algorithmes présentés au niveau complexité algorithmique et stabilité numérique, et effectuons des comparaisons avec d'autres algorithmes existants dans la littérature.

Mots clés: Matrices polynomiales, Analyse numérique, Algèbre linéaire numérique, Théorie de la commande, Structure propre, Factorisation spectrale, Logiciels de CACSD.

Abstract

In this thesis we develop new numerical algorithms for polynomial matrices. We tackle the problem of computing the eigenstructure (rank, null-space, finite and infinite structures) of a polynomial matrix and we apply the obtained results to the matrix polynomial J-spectral factorization problem. We also present some applications of these algorithms in control theory. All the new algorithms presented here are based on the computation of the constant null-spaces of block Toeplitz matrices associated to the analysed polynomial matrix. For computing these null-spaces we apply standard numerical linear algebra methods such as the singular value decomposition or the QR factorization. We also study the application of fast methods like the generalized Schur method for structured matrices. We analyze the presented algorithms in terms of algorithmic complexity and numerical stability, and we present some comparisons with others algorithms existing in the literature.

Keywords: Polynomial matrices, Numerical analysis, Numerical linear algebra, Control theory, Eigenstructure, Spectral factorization, CACSD software.

À ma femme Sofia...
To my wife Sofia...

Préface

Cette thèse s'inscrit dans la problématique du développement d'algorithmes numériques fiables pour la commande. Cela dit, et bien que nos motivations soient les problèmes numériques en commande (en particulier dans l'approche polynomiale), il ne s'agit pas d'une thèse où le lecteur trouvera des résultats nouveaux sur la théorie de la commande des systèmes. Il ne s'agit pas non plus d'une thèse d'analyse numérique pure. Nous cherchons plutôt à appliquer les résultats de l'analyse numérique et de l'algèbre linéaire numérique pour développer de nouveaux algorithmes qui résoudront, de manière fiable, des problèmes connus en commande.

Cette thèse se situe donc à la frontière entre la commande et l'analyse numérique, là où toutes les théories et toutes les approches à la commande rencontrent les mêmes problèmes liés à l'utilisation d'un système numérique (l'ordinateur par exemple): ceux dus à la précision finie des calculs arithmétiques.

Ainsi, nous supposons connus les concepts classiques d'analyse numérique. Cependant, nous rappelons quelques notions et détails quand nécessaire[1]. Dans la section 2.1.1, nous présentons de façon pratique et à l'aide d'exemples quelques idées basiques de l'analyse numérique. Bien sûr cette section n'est pas une étude exhaustive du sujet, le lecteur intéressé peut consulter l'abondante bibliographie existante. On cite ici par exemple [14, 33, 43, 80, 99] qui sont, à l'extérieur de la communauté des analystes numériques, quelques-uns des travaux les plus connus et par conséquent les plus accessibles.

Nous supposons aussi que le lecteur est familiarisé avec la théorie de la commande et ses différentes approches. De la même façon nous rappelons quelques idées, en particulier de la théorie des matrices polynomiales, quand cela peut aider à la compréhension de nos résultats. Les polynômes, les matrices polynomiales et leur algèbre est assez théorique, et il y a beaucoup de résultats dans la littérature disons mathématique, voir par exemple [27, 31].

Nous avons construit un index à la fin de ce document comme outil pour trouver rapidement les pages où les informations importantes sur les concepts clés de cette thèse sont données. La première fois qu'un de ces concepts clés apparaît dans ce document il est écrit *en italique*.

1. La plupart du temps sous forme de note de pied de page.

Dans la mesure du possible, pour un sujet donné, nous donnons les références bibliographiques originales ou celles où la théorie est compilée. Cependant, ce n'est pas la règle générale, et le lecteur découvrira qu'il y a peut-être des références à des travaux qui ne sont pas les premiers ou les originaux sur un sujet donné. Dans ce cas sachez que la seule cause de cette omission est que la référence citée est la seule portée à notre connaissance, ou en tout cas la seule que nous avons consultée.

Le sujet de cette thèse est relativement nouveau et semble intéresser de plus en plus de gens dans la communauté internationale. Alors, pour assurer sa possible diffusion dans la communauté non-francophone, l'index, ainsi que les chapitres 3, 4 et 5, qui contiennent les contributions techniques, ont été rédigés en anglais.

Des conclusions techniques sur les résultats de chaque chapitre ainsi comme les conclusions générales de cette thèse sont exposées dans le dernier chapitre en français. Un concentré des contributions et de perspectives (travail futur) est aussi présenté.

Cette thèse a été réalisée au Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS-CNRS), dans le cadre d'un doctorat à l'Institut National des Sciences Appliquées (INSA) de Toulouse, France.

Je tiens à remercier Didier Henrion pour son soutien inconditionnel comme mon directeur de recherche tout au long de ma formation doctorale, pour son interet en trouver toujours les bonnes solutions aux differents problemes que j'ai eu. Merci Didier pour ton amitié.

Je remercie les commentaires précis de Paul Van Dooren et Isham Abou-Kandil pour améliorer ce document. Je remercie aussi tous les autres membres du jury: Germain Garcia, Michael Sebek et Javier Ruiz pour son soutien. Également je remercie à la direction du LAAS-CNRS et du groupe de recherche MAC dans lequel j'ai travaillé ces 4 années.

En tant que boursier du programme de coopération CONACYT-SFERE, je remercie le soutien économique du Conseil National de la Science et de la Technologie du Mexique (CONACYT) et de la Société française d'exportation des ressources éducatives (SFERE). Je remercie aussi le soutien économique du Secrétariat d'Education Publique du Mexique (SEP).

Finalement, mais surtout, je remercie ma famille, mes parents et spécialement ma femme pour avoir pris part avec moi de cette aventure qui termine, pour son soutien et pour tous les sacrifices qu'ils ont dus réaliser par ma faute.

Juan Carlos Zúñiga Anaya.
Toulouse, France, septembre 2005.

Table des matières

Chapitre 1

Introduction

L'exécution ordonnée d'un ensemble de pas (instructions ou opérations) vers la solution d'un problème quelconque est quelque chose que l'homme a fait depuis la nuit des temps, même avant qu'il existe le mot algorithme (*algorithm*) pour définir ce processus [13]. Le mot algorithme garde une connotation mathématique qui convient parfaitement étant donnée la thématique de ce travail. Cependant, d'autres possibles synonymes comme processus, méthode, technique, procédure, règle ou recette mettent en évidence que tout le monde applique des algorithmes même inconsciemment.

Le mot algorithme est dérivé de l'arabe Al-Khwārizmī qui était le nom du mathématicien auteur du plus ancien travail connu d'algèbre. Dans son travail, Al-Khwārizmī soulignait l'importance d'appliquer une procédure méthodique pour résoudre des problèmes comme certains types d'équations quadratiques. Ce travail et beaucoup d'autres, bases de ce que l'on appelle l'algèbre, sont traduits en latin au 12ème siècle. À ce moment, et à cause de l'introduction du nouveau système de numérotation, on utilisait le mot algorithme pour nommer certaines procédures arithmétiques. Quand la numération arabe a été finalement adoptée, appliquer un algorithme signifiait simplement faire de l'algèbre. Avec le temps, l'algèbre est devenue une discipline beaucoup plus compliquée que la solution d'équations quadratiques et les algorithmes sont devenus simplement des outils pour résoudre des problèmes spécifiques de manière méthodique, systématique.

Actuellement la liste des algorithmes qui servent à résoudre un problème donné peut être, pour certains problèmes, interminable. Les algorithmes ont évolué. Une tendance naturelle est de les rendre à chaque fois plus généraux. Cette tendance est aussi encouragée par le développement de la science qui impose, à chaque fois, des problèmes plus compliqués. Prenons par exemple la solution d'un système d'équations $Ax = b$. En 1750 Cramer a formulé une procédure générale pour résoudre ce système. Avec la règle de Cramer, le problème paraissait résolu. Cependant, quand on a voulu appliquer cette règle aux nouveaux problèmes en astronomie de cette époque, où le nombre d'équations était grand, le nombre de multiplications qu'on

devait faire était énorme[1]. Face à ce problème, d'autres algorithmes, qui réduisaient le nombre d'opérations, ont été développés. C'est le cas, par exemple, de l'élimination Gaussienne en 1810.

Les algorithmes n'ont pas seulement évolué grâce aux problèmes posés par la science, mais aussi la science a avancé grâce à l'étude, l'analyse et l'application des algorithmes. Par exemple, ce n'est qu'en 1815 que Cauchy a démontré la règle de Cramer pour la première fois, en donnant l'origine à l'étude des déterminants. De manière similaire, c'est en 1850 que Sylvester a introduit le terme "matrice", des années après que ses propriétés fondamentales aient été établies [13].

Avec le développement des ordinateurs dans les dernières décennies, l'étude et l'analyse des algorithmes ont pris de nouveau beaucoup d'importance. Malgré la capacité de calcul que les ordinateurs offrent, la qualité de la solution obtenue n'est pas toujours ce que l'on peut espérer. En particulier, on peut vérifier que la plupart des algorithmes de facile exécution à la main et qui donnent de bons résultats ne deviennent plus fiables si on les codifie dans l'ordinateur. La cause: la précision finie (*finite precision*) présente dans tous les systèmes numériques. Cette précision finie limite la quantité de nombres réels que l'ordinateur peut représenter, et par conséquent des erreurs dues à l'arrondi (*rounding*) sont introduites [116].

Exemple 1.1
La solution de l'opération $0.3 - 0.2 - 0.1$ calculée par Matlab[2] [68] en double précision[3] est $-2.77... \times 10^{-17} \neq 0$. La cause est simplement qu'aucun des nombres qui intervient dans le reste ne peut être représenté exactement par l'ordinateur. Cet exemple extrait de [10] montre que les erreurs d'arrondi apparaissent dès les problèmes les plus simples. ∎

Avec la précision finie, quelques propriétés des opérations arithmétiques basiques comme la somme ou la multiplication ne sont pas vérifiées. Tel est le cas de la propriété associative de la somme comme démontré dans l'exemple suivant.

Exemple 1.2
Considérons un schéma arithmétique à double précision. L'opération $1 + 10^{20} - 10^{20}$ peut être effectuée par l'ordinateur comme $1 + (10^{20} - 10^{20})$ qui donne le résultat correct, ou bien comme $(1 + 10^{20}) - 10^{20}$ qui donne 0 car les nombres $1 + 10^{20}$ et 10^{20} sont représentés de la même manière par l'ordinateur avec la précision donnée. ∎

Intuitivement on peut deviner quelle est la façon correcte d'effectuer l'opération. Même quand il y a plusieurs opérations, on peut décider la précision avec laquelle chaque opération est effectuée en prédisant l'effet de chaque erreur dans le résultat final.

1. Environ 300 millions de multiplications pour 10 équations [13].
2. MATrix LABoratory, voir `www.themathworks.com`
3. Il s'agit du format classique de représentation de nombres à virgule flottante, avec environ 14 chiffres significatifs [32, 43, 80].

Exemple 1.3
On veut calculer les racines $x = \alpha$ et $x = \beta$ de l'équation quadratique $p(x) = ax^2 + bx + c$. À l'aide de la formule classique qu'on apprend à l'école, on peut vérifier que

$$\alpha = \frac{-b+d}{2a}, \qquad \beta = \frac{-b-d}{2a}$$

où $d = \sqrt{b^2 - 4ac}$. Supposons que $b > 0$ est tel que $b^2 \gg 4ac$. Ainsi, au moment de calculer β, on peut considérer que $d \approx b$ et choisir $\hat{\beta} = -b/a$ comme une bonne approximation de la valeur réelle de β. Par contre, au moment de calculer α on a intérêt à obtenir d avec le plus grand nombre de chiffres significatifs de telle sorte que la différence $-b + d$ ne se perde pas. ■

Malheuresement, quand des milliers d'opérations sont effectuées automatiquement par l'ordinateur, avec une même précision finie, ce contrôle manuel de l'erreur n'est plus possible. Pour l'exemple précédent, en fonction des valeurs numériques de a, b, c et de la précision utilisée, il est possible qu'une soustraction destructive [43] se produise et que la valeur obtenue de α soit très mauvaise. Voici un dernier exemple très illustratif extrait de [74].

Exemple 1.4
Supposons qu'on veuille calculer e^A, l'exponentielle de la matrice

$$A = \begin{bmatrix} -49 & 24 \\ -64 & 31 \end{bmatrix}$$

en simple précision[4]. L'algorithme classique consiste à développer la série de Taylor

$$e^A = \sum_{j=0}^{\infty} \frac{1}{j!} A^j$$

jusqu'à que les termes soient inférieurs en valeur absolue à 10^{-7}. L'algorithme est facilement programmable et le résultat est

$$e^A \approx \begin{bmatrix} -22.2588 & -1.43277 \\ -61.4993 & -3.47428 \end{bmatrix}.$$

Malheureusement le résultat correct, arrondi à 6 chiffres significatifs, est

$$e^A \approx \begin{bmatrix} -0.735759 & 0.551819 \\ -1.47152 & 1.10364 \end{bmatrix}$$

et donc même les signes de certains éléments calculés par l'ordinateur sont incorrects. En observant chaque terme de la série on peut noter que quelques termes du milieu sont proches de 10^{-7} mais de signe contraire. Par conséquent, des soustractions destructives [43] ont lieu et l'information significative est perdue. ■

4. Environ 6 chiffres significatifs [32, 43, 80].

Des erreurs numériques sont commises, donc, en effectuant des opérations arithmétiques avec précision finie ou en appliquant des algorithmes non fiables ou de façon inadéquate sur un problème quelconque. Quand ces erreurs sont commises dans le monde réel, au moment d'analyser ou de contrôler un système physique, les conséquences peuvent être catastrophiques. Le 25 Février 1991 un missile Patriot n'a pas pu intercepter un SCUD Iraquien qui a détruit des bâtiments Américains en tuant 28 soldats – la cause: une mauvaise manipulation des erreurs d'arrondi [93]. Le 4 Juin 1996, le lanceur Ariane 5 a explosé après 45 secondes de vol avec un chargement évalué à 500 millions de dollars – la cause: un dépassement (overflow[5] [32]) dans l'algorithme du système de référence d'inertie de la fusée [28].

Heureusement, la plupart des erreurs numériques sont commises en bureau d'étude: pendant la simulation et la conception des systèmes. Ces erreurs n'ont comme conséquence qu'une perte de temps. Cependant, l'analyse et la compréhension de ces erreurs peuvent nous éviter des conséquences plus graves. Donc, il est nécessaire de disposer d'une théorie permettant d'analyser et de développer des algorithmes qui prennent en considération les problèmes numériques dus à la précision finie. On parle désormais d'algorithmes numériques (*numerical algorithms*).

La communauté scientifique a accepté ce défi il y a longtemps, plusieurs disciplines scientifiques ont développé une branche d'analyse numérique (*numerical analysis*) [36]. Dans le cadre de cette thèse, l'analyse numérique est considérée comme la théorie mathématique des méthodes qui peuvent être exécutées numériquement (dans un ordinateur par exemple).

L'analyse de l'erreur arrière[6] (*backward error*) est un des éléments clés de l'analyse numérique. Le concept d'erreur arrière a été introduit pour la première fois par Von Neumann et Turing [104, 112], mais sans doute l'étude la plus complète sur l'effet des erreurs d'arrondi est celle de Wilkinson [115, 116]. L'étude de la sensibilité[7] (*sensitivity*) du problème analysé est aussi très importante pour déterminer la qualité de la solution obtenue par l'algorithme, ou erreur avant (*forward error*), étant donnée l'erreur arrière introduite. La première théorie du conditionnement[8] (*conditioning*) a été développée par Rice [89].

L'algèbre linéaire numérique (*numerical linear algebra*) est un exemple très illustratif de discipline née de l'intérêt et de la nécessité d'appliquer la théorie mathématique (l'algèbre linéaire) au monde des coefficients à précision finie où des résultats numériques sont requis. Cette discipline est consistituée d'algorithmes numériques pour résoudre des problèmes dont la formulation est parfois relativement simple. Par exemple, le problème des valeurs propres (eigenvalues), dont la théorie est connue depuis longtemps, est encore actuellement un des problèmes numériques les plus délicats [115].

5. Un nombre sur 64 bits en virgule flottante supérieur à 65536 a été transformé en un entier sur 16 bits

6. Le reflet de toutes les erreurs d'arrondi commises par un algorithme numérique sur les données d'entrée [43, 115]

7. L'évolution du résultat d'un problème suite aux changements des données d'entrée [43].

8. Mesure de la sensibilité d'un problème [43, 45]

Pendant le siècle dernier, l'algèbre linéaire numérique a été très développée. La production d'algorithmes numériques a été très vaste. En 2000 l'IEEE Computer Society et l'Institut Américain de Physique ont compilé une liste des 10 meilleurs algorithmes du siècle [16, 22]. Parmi ces 10 algorithmes, 4 sont des algorithmes numériques d'algèbre linéaire ou appliqués à l'algèbre linéaire:

- Les méthodes itératives sur les sous-espaces de Krylov (méthode de Lanczos, méthode du gradient, etc) en 1950.
- L'approche de factorisation des matrices de Alston Householder en 1951.
- L'algorithme de compilation pour FORTRAN en 1957.
- L'algorithme QR de J.G.F. Francis en 1959.

Sans doute un des algorithmes les plus importants est l'algorithme de compilation de FORTRAN. Avec le langage de programmation FORTRAN [9] les scientifiques et ingénieurs ont pu finalement dire à l'ordinateur exactement ce qu'il doit faire sans passer des heures en programmant en langage machine. Le calcul scientifique (*scientific computing*) est né sur la base de l'algèbre linéaire numérique et la programmation en FORTRAN [85].

Actuellement les bibliothèques comme LAPACK [10], successeur d'EISPACK et de LINPACK, contiennent de nombreuses routines programmées en FORTRAN pour résoudre des problèmes divers de l'algèbre linéaire comme par exemple: les valeurs propres, la factorisation SVD, QR, etc. [3]. La bibliothèque LAPACK est basée sur des sous-programmes en FORTRAN pour effectuer des opérations basiques sur les vecteurs et les matrices. Cet ensemble de sous-programmes est appelé BLAS [11] [62].

La performance de LAPACK et BLAS est remarquable. Grâce à cette performance, ces bibliothèques d'algèbre linéaire numérique sont utilisées comme des "boîtes noires" à la base de logiciels de plus haut niveau de programmation comme par exemple Matlab ou Scilab [12] [91]. Une autre caractéristique très importante de LAPACK et de BLAS est leur portabilité, c'est-à-dire leur indépendance de la plateforme ou du système opérationnel utilisé. Cette portabilité a été favorisée par la création, en 1985, du standard ANSI/IEEE 754 pour les modèles arithmétiques en virgule flottante (*floating point arithmetic*) [73].

Actuellement on trouve LAPACK et BLAS à la base de logiciels scientifiques de tous types et pour des applications variées. En particulier, ces bibliothèques sont la base de beaucoup de logiciels pour la conception et la solution de problèmes en commande, constituant les logiciels de *CACSD (Computer Aided Control System Design)*. Dans le cadre d'IEEE, la Control System Society anime depuis 1982 un comité technique sur les logiciels de CACSD [13] [41]. On approfondit ce sujet dans le chapitre suivant.

9. Langage continuellement actualisé, voir `www.fortran.com`
10. Linear Algebra PACKage, voir `www.netlib.org/lapack`.
11. Basic Linear Algebra Sub-programmes, voir `www.netlib.org/blas`.
12. SCIentific LABoratory, voir `www.inria.fr/scilab`.
13. IEEE CSS TC on CACSD, voir `www.laas.fr/cacsd`

Un phénomène qui a aussi contribué énormément au développement de l'algèbre linéaire numérique sont les relations symbiotiques que cette discipline a eu avec d'autres disciplines en science et en ingénierie. Un exemple très clair est la théorie de la commande des systèmes linéaires représentés sous la forme matricielle (espace d'état). D'une part la théorie des systèmes est pleine de problèmes d'algèbre linéaire numérique, et d'autre part beaucoup d'algorithmes de l'algèbre linéaire numérique trouvent des applications intéressantes dans la théorie des systèmes. Cette interaction entre algèbre linéaire numérique et commande a été très fructueuse. L'approche espace d'état (*state space approach*) [50, 90], suite aux travaux de Rudolf Kalman dans les années 1960, est devenue l'approche largement préférée en ce qui concerne l'obtention de résultats numériques. Beaucoup de logiciels de CACSD ont été développés dans ce cadre. On approfondit ce sujet dans le chapitre suivant.

Par contre, d'autres disciplines mathématiques ont moins développé leurs côtés numériques. C'est le cas de l'algèbre non linéaire et de la géométrie algébrique qui sont restés assez théoriques [98]. Pourtant, actuellement il est devenu de plus en plus nécessaire d'étendre cette théorie à des résultats numériques. Les polynômes par exemple sont devenus des outils naturels pour modéliser les systèmes dynamiques, qui, dans le monde réel, ont des coefficients numériques avec une précision limitée.

Grâce à la capacité des ordinateurs modernes[14] le calcul symbolique (*symbolic computing*) est devenu possible et l'algèbre calculatoire (*computer algebra*) a été développée comme outil informatique pour faire des mathématiques pures avec l'ordinateur. Quelques exemples de logiciels qui permettent le calcul symbolique sont: MAPLE[15] [113] et MATHEMATICA[16] [117]. Cependant, les résultats obtenus par le calcul symbolique sont en général inadéquats pour les problèmes réels. Par exemple, le résultat d'un système d'équations à coefficients numériques peut être donné comme des fractions d'entiers avec des centaines de chiffres. Le passage au monde numérique n'est pas toujours adéquat.

Des efforts pour raccourcir le chemin entre l'algèbre calculatoire et l'analyse numérique sont faits ces dernières années. Différents groupes de recherche dans les universités ont commencé à développer des logiciels numériques pour les problèmes sur les polynômes[17]. Quelques bibliothèques, comme par exemple NAG[18], ont été développées. Ces bibliothèques incluent plusieurs routines pour résoudre des problèmes numériques qui ne sont pas de l'algèbre linéaire, comme par exemple: l'extraction de racines de polynômes, les équations différentielles, l'interpolation, l'optimisation, etc. Des logiciels comme MAPLE basent leurs capacités numériques sur les bibliothèques NAG. Malheureusement, les bibliothèques NAG ne sont pas de libre distribution (contrairement à LAPACK ou BLAS) et il reste encore beaucoup de travail à faire, tant au niveau théorique que pratique, pour pouvoir parler d'une algèbre non linéaire numérique (*numerical non-linear algebra*).

14. Surtout au niveau de mémoire de stockage.
15. MAthematical PLEasure, voir `www.maplesoft.com`.
16. Voir `www.wolfram.com`.
17. Voir par exemple `www.dm.unipi.it/~ananum/polynomial.html`. [8]
18. Numerical Algorithms Group, voir `www.nag.co.uk`.

Par rapport à la théorie de la commande des systèmes, ces dernières décennies d'autres approches ont été développées comme alternatives à l'approche espace d'état : l'approche polynomiale (*polynomial approach*) [58] et l'approche comportementale (*behavioural approach*) [83]. Avec ces approches les systèmes sont représentés à l'aide de polynômes et de matrices polynomiales. Il devient alors nécessaire de réaliser plusieurs tâches numériques comme la manipulation des polynômes et des matrices polynomiales, la solution d'équations polynomiales et l'extraction de propriétés structurelles.

Malheureusement, et malgré le fait que l'approche polynomiale est actuellement un des plus puissants outils théoriques pour la commande des systèmes (voir par exemple [48]), au niveau des logiciels de CACSD son développement est beaucoup plus faible que celui de l'approche espace d'état. La raison principale, comme on le verra dans le prochain chapitre, est le manque d'une théorie numérique consolidée pour les polynômes et les matrices polynomiales, et par conséquent un manque de logiciels de base comme LAPACK pour l'algèbre linéaire numérique.

Récemment, sous le pseudonyme d'algèbre polynomiale numérique (*numerical polynomial algebra*), la théorie de l'analyse numérique a finalement été adaptée aux polynômes par Hans Stetter [98]. Il s'agit d'une des premières contributions vers la consolidation d'une algèbre non linéaire numérique. Dans la communauté de la commande de systèmes, beaucoup d'enthousiastes des méthodes polynomiales ont fait aussi des efforts pour combler la manque de logiciels de CACSD pour l'approche polynomiale (voir le chapitre suivant). Plusieurs algorithmes modernes pour les polynômes et les matrices polynomiales sont maintenant basés sur des opérations stables de l'algèbre linéaire numérique, voir [5, 38, 107] par exemple. Des résultats récents comme [97, 119], bien qu'ils ne soient pas encore très diffusés et appliqués, prédisent que beaucoup de problèmes polynomiaux mal conditionnés ou mal posés peuvent être résolus de manière satisfaisante dans le sens de l'analyse numérique et de l'algèbre polynomiale numérique. Ainsi nous croyons qu'il n'est qu'une question de temps pour qu'on puisse appliquer la théorie de l'analyse numérique aux problèmes polynomiaux en commande et pour que le développement de logiciels de CACSD pour l'approche polynomiale soit comparable à celui de l'approche espace d'état.

1.1 Objectif de cette thèse

L'objectif de cette thèse est de contribuer au développement d'algorithmes numériques fiables pour les matrices polynomiales en ayant comme motivation les problèmes de commande. Le contenu de la thèse est résumé par les points suivants:

- Etablissement de l'état de l'art des logiciels de CACSD tant pour l'approche polynomiale que pour l'approche espace d'état. On montre comment l'algèbre linéaire numérique aide à résoudre plusieurs problèmes numériques en commande.
- Développement de nouveaux algorithmes numériques pour l'obtention de la structure propre des matrices polynomiales et pour la solution du problème

de factorisation J-spectrale. On montre que ces algorithmes sont entièrement basés sur des méthodes stables de l'algèbre linéaire numérique.
- Analyse de la complexité algorithmique et de la stabilité numérique de chaque algorithme. Cette analyse est basée sur la stabilité des méthodes numériques de l'algèbre linéaire utilisées. Par contre, on montre que l'étude de la nature (sensibilité, conditionnement, etc.) des problèmes analysés est plus compliquée et requiert la théorie de l'algèbre polynomiale numérique[19].
- Plusieurs exemples académiques mais aussi d'applications en commande pour illustrer les résultats obtenus et les algorithmes développés.

1.2 Organisation des chapitres

Ce document est organisé de la manière suivante:

Chapitre 1: Cette introduction.

Chapitre 2: Les différentes approches à la commande des systèmes sont présentées. Ensuite on établit un état de l'art des logiciels de CACSD pour l'approche espace d'état et pour l'approche polynomiale. On analyse les problèmes numériques associés aux matrices polynomiales et on introduit les méthodes qui servent à traiter ces problèmes. Dans ce chapitre on montre comme certains problèmes sur les matrices polynomiales peuvent être résolu en résolvant un problème équivalent sur les matrices constantes. On présente aussi les différentes approches à cette procédure. À l'aide des différents exemples présentés dans ce chapitre on rappelle les concepts basiques de l'analyse numérique. L'étude formelle de ces concepts n'est pas un objectif de cette thèse, donc on invite les lecteurs intéressés à consulter l'abondante littérature sur le sujet.

Chapitre 3: Un nouvel algorithme pour calculer la structure finie et infinie d'une matrice polynomiale est présenté. On suppose que les zéros finis de la matrice analysée sont donnés, cependant on consacre une petite section à la problématique du calcul des zéros et aux différentes méthodes existantes. On introduit aussi un peu de théorie sur les matrices polynomiales et on montre comment cette théorie peut être comprise d'une manière algorithmique. En particulier, on donne des conditions pûrement algébriques (sur des matrices constantes) pour l'absence de zéros à l'infini d'une matrice polynomiale. L'analyse de la complexité et de la stabilité de l'algorithme est développée ainsi que des comparaisons avec d'autres algorithmes existants. Quelques exemples d'applications à la commande sont aussi présentés.

Chapitre 4: Un nouvel algorithme pour l'obtention de l'espace nul d'une matrice polynomiale est décrit. L'analyse de la complexité et de la stabilité de

19. Cette étude dépasse les objectifs de cette thèse et seulement des idées de travaux futurs sont présentées

l'algorithme est présentée ainsi que des comparaisons avec d'autres algorithmes existants. On illustre les résultats avec quelques exemples d'applications à la commande.

Chapitre 5: Le problème de la factorisation des matrices polynomiales est introduit. On montre comment les algorithmes présentés dans les chapitres précédents peuvent être appliqués pour factoriser la matrice polynomiale analysée. On montre qu'on peut extraire des facteurs qui ne contiennent qu'une partie de la structure propre de la matrice polynomiale. Finalement, on présente un nouvel algorithme pour la factorisation J-spectrale des matrices polynomiales ainsi que des comparaisons avec d'autres algorithmes existants.

Chapitre 6: Dans ce chapitre on trace les conclusions générales, un résumé des contributions de la thèse et les perspectives et idées de travaux futurs.

Chapitre 2

Méthodes numériques et logiciels de CACSD

Dans ce chapitre on présente les différentes approches à la commande de systèmes linéaires. Ensuite on présente quelques caractéristiques des logiciels de CACSD (Computer-Aided Control System Design) existant pour l'approche espace d'état et pour l'approche polynomiale. On analyse les problèmes numériques associés aux matrices polynomiales et on introduit les méthodes qui servent à traiter ces problèmes. Il faut rappeler que l'analyse numérique est le fil conducteur de la discussion, mais ce n'est pas le thème central ni l'objectif de la thèse. Les concepts généraux de cette théorie sont rappelés dans les différents exemples présentés dans ce chapitre, mais les lecteurs intéressés par une étude plus systématique, formelle et approfondie devront consulter l'abondante littérature sur le sujet.

Le processus de la commande d'un système dynamique commence par sa modélisation en termes d'équations mathématiques. En particulier, dans le cas des systèmes linéaires invariants dans le temps, on a un ensemble de p équations différentielles du type

$$\begin{aligned} &a_{1d_1} y_1^{(d_1)}(t) + \cdots + a_{10} y_1(t) + a_1 + \cdots + a_{pd_P} y_p^{(d_p)}(t) + \cdots + a_{p0} y_p(t) + a_p = \\ &b_{1f_1} u_1^{(f_1)}(t) + \cdots + b_{10} u_1(t) + b_1 + \cdots + b_{mf_m} u_m^{(f_m)}(t) + \cdots + b_{m0} u_m(t) + b_m \end{aligned} \quad (2.1)$$

où $\alpha^{(q)}(t)$ représente la q-ième dérivée du signal scalaire $\alpha(t)$ par rapport au temps. Les paramètres a_{ij} et b_{kl} pour $i = 1{:}p$, $j = 0{:}d_i$, $k = 1{:}m$ et $l = 0{:}f_k$ sont constants. Par convention, toutes les variables sur lesquelles on a un contrôle et qui serviront à commander le système sont appelées les entrées $u_i(t)$, $i = 1{:}m$. D'autre part, toutes les variables sur lesquelles on observe l'effet de la commande sont appelées les sorties $y_i(t)$, $i = 1{:}p$.

Il existe différentes approches pour interagir avec ce modèle mathématique. Avec chaque approche, le modèle est représenté ou reformulé d'une façon particulière où les différentes propriétés du système sont plus ou moins dévoilées. On peut par exemple écrire chaque équation de type (2.1) comme un ensemble d'équations différentielles d'ordre 1 à l'aide de l'inclusion de variables d'état (*state variables*) [50]. Après des

manipulations algébriques, le modèle du système peut être écrit de la manière suivante:

$$\begin{aligned} x^{(1)}(t) &= Ax(t) + Bu(t), \\ y(t) &= Cx(t) + Du(t). \end{aligned} \tag{2.2}$$

Le vecteur $x(t) \in \mathbb{R}^n$ est le vecteur des variables d'état et sert donc à relier le vecteur d'entrées $u(t) \in \mathbb{R}^m$ et le vecteur de sorties $y(t) \in \mathbb{R}^p$. Les matrices $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$ et $D \in \mathbb{R}^{p\times m}$ sont constantes. La représentation (2.2) est utilisée par l'approche espace d'état [50, 90]. Notons qu'en appliquant la transformée de Laplace sur la représentation (2.2) on peut éliminer $x(t)$ et en déduire la fonction de transfert $T(s)$ du système [50]

$$y(s) = [C(sI - A)^{-1}B + D]u(s) = T(s)u(s)$$

dans l'hypothèse de conditions initiales nulles.

Une autre manière de réécrire l'ensemble des p équations de type (2.1) qui représentent le système dynamique est la forme matricielle. En regroupant toutes les entrées et sorties dans les vecteurs correspondants u et y on obtient:

$$D(s)y(s) = N(s)u(s). \tag{2.3}$$

Les matrices $D(s)$ et $N(s)$ sont des matrices polynomiales (*polynomial matrices*) en la variable s, de dimensions $p \times p$ et $p \times m$ respectivement. Rapellons que dans le cas continu, la variable s est l'opérateur de Laplace. Si le système est à temps discret, alors la variable devient l'opérateur de décalage.

Étant donné que la fonction de transfert du système est unique [50], on vérifie que

$$T(s) = D^{-1}(s)N(s) = C(sI - A)^{-1}B + D.$$

La représentation (2.3) est utilisée par l'approche polynomiale [58] ou encore par l'approche comportementale [83].

Chaque représentation a ses points forts et ses points faibles, et implique une façon différente d'analyser le même système. Par conséquent chaque approche à la commande des systèmes a aussi ses particularités et utilise ses propres méthodes et outils mathématiques. Une discussion des avantages ou désavantages de chaque approche impliquerait beaucoup d'efforts pour n'en tirer aucune conclusion. En fait, le vrai avantage consiste à disposer de plusieurs façons différentes de résoudre un problème. Pour certains problèmes l'approche espace d'état sera plus efficace, pour d'autres problèmes l'approche polynomiale ou l'approche comportementale seront plus adéquates. Actuellement, au niveau théorique, quelle approche utiliser reste simplement une question de préférence personnelle. Cependant, au niveau des logiciels de CACSD, l'approche espace d'état est beaucoup plus développée. On illustre ceci dans les sections suivantes.

2.1 Approche espace d'état

On considère un système donné sous la forme (2.2). L'objet mathématique du modèle sont des matrices constantes à valeurs réelles. Par conséquent, l'algèbre linéaire et l'algèbre linéaire numérique sont les outils mathématiques de base pour l'analyse et la synthèse de la commande des systèmes décrits par l'approche espace d'état. L'interaction entre algèbre linéaire et commande a été très fructueuse ces dernières années. Beaucoup de problèmes numériques en commande sont résolus de manière satisfaisante grâce à l'application des méthodes numériques de l'algèbre linéaire. L'utilisation des librairies numériques comme LAPACK ou BLAS a permis la conception de centaines de logiciels de CACSD. Plusieurs logiciels furent initialement présentés dans un numéro spécial de la revue IEEE Control Systems Magazine en 1982 [41]. Actuellement, certains de ces logiciels sont mis à jour et de nouveaux apparaissent. On peut mentionner, par exemple, la *Control System Toolbox* pour Matlab[1] et la bibliothèque *Slicot*[2] qui sont deux outils assez connus et utilisés.

Les critères pour juger de la supériorité d'un logiciel de CACSD ne sont pas établis d'une façon unifiée et acceptée par tout le monde. Chaque concepteur ou utilisateur souligne certains critères plus que d'autres et base ses points forts sur différentes caractéristiques. Cependant, quelques réflexions générales peuvent se faire à ce propos:

- Les logiciels de CACSD peuvent être regroupés en deux grandes catégories. Le niveau inférieur contient les logiciels numériques comme les bibliothèques LAPACK, BLAS, etc. et les fonctions basiques programmées avec un langage générant du code machine (FORTRAN est le plus utilisé). Le niveau supérieur contient les procédures d'analyse et de synthèse de commande mises en ouvre en appelant des fonctions de plus bas niveau. Ce niveau supérieur constitue surtout l'interface qui permet à l'utilisateur d'appliquer les outils numériques pour résoudre son problème de commande.
- Aucun logiciel de CACSD ne peut être plus performant que le logiciel numérique sur lequel il est construit. Dans ce sens, la meilleure précision (*accuracy*) des opérations qu'on peut espérer provient des routines de niveau inférieur. Cependant, cette précision n'est pas toujours obtenue et cela dépend, entre autres, de la façon et du moment auquel le logiciel numérique de base est utilisé: si la plupart du calcul numérique est mis en oeuvre à bas niveau, le temps d'exécution sera sûrement plus court et il y aura plus de chances pour qu'une précision satisfaisante soit obtenue.
- L'interface est un aspect qui peut différencier un logiciel CACSD d'un autre et qui est particulièrement importante pour l'utilisateur qui n'est pas familiarisé avec les problèmes et les algorithmes numériques. Certains logiciels de CACSD fournissent leur propre interface [41], d'autres logiciels utilisent Matlab pour créer cette interface avec l'utilisateur. En tout cas, notons qu'une bonne interface rend plus facile l'utilisation mais peut aussi compromettre largement le

1. Voir `www.mathworks.com/products/control`.
2. Créée par The Numerics in Control Network NICONET. Voir `www.win.tue.nl/niconet`.

temps d'exécution et parfois même la précision.

L'utilisation de logiciels de CACSD a permis d'étendre l'utilisation de l'approche espace d'état dans la communauté. Ainsi, beaucoup de gens ont pu appliquer de complexes théories de commande en obtenant des résultats numériques mais sans s'occuper de la problématique numérique et algorithmique. Indiscutablement c'est très positif. Cependant, il faut rappeler que derrière tout logiciel il y a beaucoup de travail et de développement. D'une part il y a la théorie des mathématiques appliquées (l'algèbre linéaire numérique) qui n'est pas propre à l'approche espace d'état, et d'une autre part l'adaptation des résultats et algorithmes classiques en commande pour prendre en compte les problèmes numériques.

2.1.1 Application des logiciels de CACSD

Par la suite on illustre l'application des algorithmes classiques en commande, en utilisant les outils de l'algèbre linéaire numérique, pour faire face aux problèmes numériques et obtenir des résultats satisfaisants. Les algorithmes qu'on présente ici ne sont pas nouveaux, ils sont décrits à titre illustratif. Les résultats originaux peuvent se trouver, par exemple dans [45, 50, 61, 82, 108] et leurs références. Un deuxième objectif de cette section est de présenter de manière pratique les concepts d'analyse numérique qu'on utilisera dans le reste de cette thèse. Pour un approfondissement sur ces concepts voir par exemple [33, 43, 99, 115, 116].

Pour atteindre les objectifs de cette section, on étudie un des problèmes classiques en commande: le placement de pôles. Etant donné un système sous la forme (2.2), l'objectif est de trouver une commande linéaire statique $u = Fx$ telle que le système en boucle fermée ait des pôles donnés, c'est-à-dire, telle que la matrice $A + BF$ ait le polynôme caractéristique désiré

$$\det(sI - A - BF) = s^n + f_{n-1}s^{n-1} + \cdots + f_0 = f(s).$$

Algorithme classique et ses problèmes numériques

Théoriquement ce problème n'est pas compliqué. Une manière méthodique de le résoudre consiste à exprimer la forme canonique

$$\bar{A} = T^{-1}AT = \begin{bmatrix} -a_{n-1} & -a_{n-1} & \cdots & -a_0 \\ 1 & 0 & & \\ & \ddots & \ddots & \\ & & 1 & 0 \end{bmatrix}, \qquad \bar{B} = T^{-1}B = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{2.4}$$

avec la transformation linéaire

$$T = \mathcal{C}\mathcal{T} = \begin{bmatrix} B & AB & \cdots A^{n-1} \end{bmatrix} \begin{bmatrix} 1 & a_{n-1} & \cdots & a_1 \\ & 1 & \ddots & \vdots \\ & & \ddots & a_{n-1} \\ & & & 1 \end{bmatrix}, \tag{2.5}$$

et de choisir

$$\bar{F} = FT = \left[\begin{array}{ccc} a_{n-1} - f_{n-1} & \cdots & a_0 - f_0 \end{array} \right].$$

De cette façon, notons que

$$\det(sI - \bar{A} - \bar{B}\bar{F}) = f(s)$$

et, étant donné que T est une transformation de similarité,

$$\det(sI - A - BF) = f(s).$$

Ainsi, algorithmiquement, le problème du placement des pôles est réduit à la solution du système d'équations $FT = \bar{F}$ où T et $\bar{F}$ sont donnés et F est à déterminer, c'est la formule d'Ackermann [2]. Donc les problèmes numériques que l'on peut rencontrer sont ceux d'un système d'équations du type $My = b$, où M dénote à présent la matrice T, et $b = \bar{F}$.

Soit $M \in \mathbb{R}^{n \times n}$ et $b \in \mathbb{R}^n$. Le problème qui consiste à déterminer $y = f(x) = f(M, b) \in \mathbb{R}^n$ est un problème mal posé (*ill-posed*) au sens classique de Hadamard [35]. En fait y ne dépend pas de manière continue des données x. Il existe des points de discontinuité ou singularités (*singularity*). Pour le cas carré, il s'agit de toutes les matrices M de rang plus petit que n telles que le système n'a pas de solution ou que la solution n'est pas unique. Donc, pour que le problème ait un sens, c'est-à-dire qu'il soit bien posé (*well-posed*), on restreint l'espace des données d'entrée aux points réguliers uniquement. La question qu'on se pose alors est comment mesurer la distance à la singularité d'un point x? L'analyse de la sensibilité peut donner une réponse à ces questions.

La sensibilité de y par rapport aux perturbations sur x est mesurée par le conditionnement $\kappa(x)$ (conditioning). Intuitivement on peut conclure que si x est proche d'une singularité alors $\kappa(x)$ est grand car, dans ce cas, un petit changement dans x peut faire que y n'existe plus. Donc, $1/\kappa(x)$ est une mesure de la distance à la singularité. Si $\kappa(x)$ est grand on dit que le problème pour x donné est mal conditionné (*ill-conditioned*) ou que le point x est mal conditionné. Par contre, le problème est bien conditionné (*well-conditioned*) si $\kappa(x)$ est petit.

La connaissance du conditionnement d'un problème est importante car elle peut nous donner un indice sur la qualité de la solution calculée. Une fois qu'un problème $y = f(x)$ et son conditionnement $\kappa(x)$ sont bien définis, on peut estimer la magnitude de l'erreur en la solution calculée $\hat{y}$, appelée erreur en avant (forward error) grâce à l'expression suivante:

$$\|y - \hat{y}\| \leq \kappa(x)\|e\| \tag{2.6}$$

où e, l'erreur en arrière (backward error) est l'erreur introduite par l'algorithme appliqué sur les données d'entrée. Le conditionnement d'un même problème peut être différent pour différents types de perturbations. Alors, en fonction du type de perturbation introduit par l'algorithme, on devra utiliser la définition de conditionnement adéquate pour estimer la qualité de la solution obtenue.

Pour illustrer ces idées on retourne au problème de placement de pôles et la solution de $My = b$, en supposant M régulière, c'est-à-dire inversible. Le conditionnement de la matrice M pour le problème $My = b$ est donné par

$$\kappa_p(M) = \|M^{-1}\|_p \|M\|_p \tag{2.7}$$

pour des perturbations arbitraires limitées en norme p, ou par

$$\kappa_*(M) = \||M^{-1}||M|\|_\infty \tag{2.8}$$

pour le cas où chaque élément de M et/ou b est perturbé[3].

Exemple 2.1
Pour résoudre un problème de placement de pôles, on veut calculer la solution y du système $My = b$ avec

$$M = \begin{bmatrix} -100 & 0 & 0 \\ 0 & 85 & -52 \\ 0 & -0.003 & -0.003 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ 85 \\ -0.003 \end{bmatrix}.$$

Supposons que les calculs soient effectués dans un système arithmétique à virgule flottante à 16 bits de précision, c'est-à-dire, avec une unité d'arrondi $\mu = 0.5\varepsilon \approx 7.63 \times 10^{-6}$. Le paramètre ε est la distance entre le nombre 1 et le nombre supérieur le plus proche. Dans cette thèse, ε sera dénommé précision machine (*machine precision*). Alors, pour des raisons pratiques, on considère les résultats donnés par Matlab (avec une unité d'arrondi $\mu_m = 0.5\texttt{eps} \approx 1.11 \times 10^{-16}$) comme exacts. Considérons qu'on applique un algorithme stable[4] (*backward stable*) c'est-à-dire que la solution calculée $\hat{y}$ est la solution exacte du système légèrement perturbé

$$(M + \Delta)\hat{y} = b,$$

où Δ est une perturbation arbitraire (erreur arrière) telle que

$$\|\Delta\|_\infty \leq O(\mu)\|M\|_\infty.$$

Pour ce type de perturbation, la matrice M est très mal conditionnée avec $\kappa_\infty(M) = 28334.333...$ Alors, même si la méthode numérique utilisée pour résoudre le système $My = b$ est stable, étant donnée l'équation (2.6), il peut arriver que l'erreur avant $\|y - \hat{y}\|$ soit grande. Soit

$$\Delta = \text{rand}(3,3)\|M\|_\infty, \qquad \|\Delta\|_\infty = 0.0017... \leq 0.002... = \mu\|M\|_\infty$$

l'erreur arrière introduite par l'algorithme. Alors, $\hat{y}$ est la solution exacte

$$\hat{y} = (M + \Delta)^{-1}b = \begin{bmatrix} 1.499... \times 10^{-6} \\ 1.111... \\ 0.182... \end{bmatrix}.$$

3. En général M^{-1} n'est pas connu ou il est coûteux de l'obtenir en termes calculatoires. Alors l'estimation du conditionnement par des méthodes approximatives est nécessaire [43]. LAPACK contient de très bonnes méthodes pour estimer le conditionnement d'une matrice [3].

4. Plusieurs algorithmes stables pour la solution des systèmes d'équations sont présentés dans [33] par exemple.

La borne supérieure de l'erreur avant est donc $\kappa_\infty(M)\|\Delta\|_\infty = 49.213...$ ce qui n'incite pas à faire confiance à la qualité du résultat. Cependant, notons que cette borne est en fait assez pessimiste car le résultat n'est pas si mauvais que ça

$$\|y - \hat{y}\|_\infty = 0.182... \ll 49.213...$$

Maintenant supposons que

$$\bar{M} = PM = \begin{bmatrix} -100 & 0 & 0 \\ 0 & 85 & -52 \\ 0 & -72.733151 & -72.7331151 \end{bmatrix}, \quad \bar{b} = Pb = \begin{bmatrix} 0 \\ 85 \\ -72.7331151 \end{bmatrix}.$$

La matrice $\bar{M}$ est bien conditionnée, $\kappa_\infty(\bar{M}) = 2.302...$ Soit

$$\bar{\Delta} = \Delta\|\bar{M}\|_\infty\|M\|_\infty^{-1} \qquad \|\bar{\Delta}\|_\infty = 0.0018... \leq 0.0022... = \mu\|\bar{M}\|_\infty$$

l'erreur arrière introduite par l'algorithme. Alors, $\hat{y}$ est la solution exacte

$$\hat{\bar{y}} = (\bar{M} + \bar{\Delta})^{-1}\bar{b} = \begin{bmatrix} 1.212... \times 10^{-7} \\ 0.999... \\ 1.121... \times 10^{-5} \end{bmatrix}.$$

Dans ce cas, le borne supérieure de l'erreur avant, donnée par $\kappa_\infty(\bar{M})\|\bar{\Delta}\|_\infty = 0.004...$, est beaucoup plus optimiste. En plus, notons que $\hat{\bar{y}}$ est aussi une solution de $My = b$ mais, par conséquent, plus exacte:

$$\|y - \hat{\bar{y}}\|_\infty = 1.121... \times 10^{-5} \leq 0.004...$$

La matrice P est dénommée matrice de *pré-conditionnement* ou *scaling*. ∎

Pré-conditionnement ou scaling

Les techniques de scaling sont couramment utilisées en algèbre linéaire numérique. Actuellement, plusieurs fonctions de Matlab ou LAPACK appliquent ces techniques, même si cela passe inaperçu pour l'utilisateur standard. Par exemple: la fonction `eig` de Matlab calcule, sauf si l'on indique le contraire, les valeurs propres d'une matrice après avoir appliqué un scaling pour équilibrer les normes de ses colonnes et de ses lignes [79, 81]. La fonction `ss` de la Control System Toolbox renvoie un système (2.2) sous représentation d'état où la matrice A est automatiquement équilibrée de façon à ce qu'elle soit mieux conditionnée pour le problème des valeurs propres. Dans l'exemple 2.1 le scaling a servi d'une part à trouver une meilleure borne pour l'erreur avant, mais aussi on peut vérifier que la solution obtenue après le scaling est plus exacte[5].

Une manière de définir le conditionnement de M pour le problème $My = b$, également très utile, consiste à choisir la norme Euclidienne dans la relation (2.7):

$$\kappa_2(M) = \|M^{-1}\|_2\|M\|_2 = \frac{\sigma_{\max}(M)}{\sigma_{\min}(M)}$$

5. Ils existe d'autres techniques pour améliorer la solution calculée d'un système d'équations. Il s'agit des techniques d'affinement itératif (iterative refinement) [43, 99].

où $\sigma_{\max}(M)$, $\sigma_{\min}(M)$ sont la plus grande et la plus petite valeur singulière de M respectivement. Cette relation met en évidence que $1/\kappa_2(M)$ est bien une mesure de la distance à la singularité du problème $My = b$. Autrement dit, si $\kappa_2(M)$ est grand, M est peut être singulière. Pour une unité d'arrondi $\mu = 0.5\varepsilon$, la matrice M est singulière si $\kappa_2(M) > 1/n\varepsilon$. De manière équivalente, l'approximation numérique du rang[6] de M est égal au nombre de ses valeurs singulières supérieures à $n\varepsilon\sigma_{\max}$. Les valeurs singulières de M pour l'Exemple 2.1 sont $\sigma_{\max} = 100$, $\sigma = 99.644...$ et $\sigma_{\min} = 0.0041...$ Alors, le rang de cette matrice, pour la précision utilisée, est seulement deux car $\sigma_{\min} < 3\varepsilon\sigma_{\max} = .0045...$ Par contre, après le scaling, $\sigma_{\min}(PM) = 96.719... > 0.019... = 3\mu\sigma_{\max}(PM)$ ce qui rend la matrice $\bar{M}$ régulière.

Par rapport au problème de placement de pôles, la restriction que la matrice T donnée par (2.5) soit de rang plein est naturelle car elle implique tout simplement que le système est commandable. Cependant, tester la commandabilité du système en observant les valeurs singulières de la matrice T ou $\mathcal{C}$ n'est pas une méthode fiable. Etant donnée sa construction, on peut attendre que les normes des colonnes de la matrice de commandabilité $\mathcal{C}$ soient très déséquilibrées pour une matrice d'état arbitraire de dimension grande. Par conséquent, on peut attendre un grand écart entre $\sigma_{\max}(\mathcal{C})$ et $\sigma_{\min}(\mathcal{C})$, c'est-à-dire, que la matrice $\mathcal{C}$ soit singulière pour la précision donnée.

Exemple 2.2
Soit un système sous la forme (2.2) avec $n = 7$, $m = 1$ et les éléments de A,b choisis aléatoirement uniformément entre 0 et 1. Soit

$$\mathcal{C} = \begin{bmatrix} b & Ab & \cdots & A^6 b \end{bmatrix}$$

sa matrice de commandabilité. On considère toujours une unité d'arrondi $\mu = 0.5\varepsilon \approx 7.63 \times 10^{-6}$. Les valeurs singulières de $\mathcal{C}$ sont:

$$\sigma_1 = 2180.718..., \quad \sigma_2 = 0.886..., \quad \sigma_3 = 0.300..., \quad \cdots \quad \sigma_7 = 0.000...$$

Alors, pour la précision donnée, $\mathcal{C}$ est singulière car $\kappa_2(\mathcal{C}) = 2462878.095... > 1/7\varepsilon = 9362.285...$ En fait, notons qu'à partir de $i = 4$, $\sigma_i < 7\varepsilon\sigma_1$. Donc l'approximation numérique du rang de $\mathcal{C}$ est 3. ∎

On a vu que le scaling peut servir à régulariser notre problème $FT = \bar{F}$ pour trouver une solution plus ou moins exacte, cf. l'Exemple 2.1. Cependant, il faut considérer les remarques suivantes:

– Dans l'Exemple 2.1 il est relativement facile de déterminer la matrice de scaling P. En fait, il suffit d'observer que la décomposition en valeurs singulières de M est donnée par $M = I\Sigma V$. Alors, si on veut modifier la magnitude de $\sigma_{\min}(M)$ de telle sorte que $\sigma_{\min}(M) \approx \sigma_{\max}(M)$, il suffit de multiplier la dernière ligne de M par une constante adéquate. Dans ce cas $P = \text{diag}\{1, 1, \kappa_2(M)\}$. Malheureusement, en général il est beaucoup plus compliqué de déterminer la matrice P.

6. Voir la documentation de la fonction `rank` de Matlab, et la section 2.3.1 de cette thèse.

– En plus, il existe une limite pour les effets du scaling: on peut vérifier que, avec un scaling optimal, $\kappa_\infty(PM) = \kappa_*(M)$, où le conditionnement est défini en (2.8). Noter que la valeur de $\kappa_*(M)$ est indépendante du scaling, c'est-à-dire, $\kappa_*(M) = \kappa_*(PM)$, cf. [43, Chapitre 7]. D'après les propriétés des normes, on peut vérifier que si $\kappa_*(M) > 1/\varepsilon$, alors la matrice PM pour toute P diagonale sera toujours singulière pour la précision donnée. Dans l'exemple 2.2 $\kappa_*(\mathcal{C}) = 2168168.553... > 65536 = 1/\varepsilon$ donc on ne pourra pas régulariser le problème avec un scaling même si on arrive à trouver la matrice P optimale.

Donc notons que malgré les techniques de scaling et l'existence d'algorithmes stables pour la solution du système $FT = \bar{F}$, une solution satisfaisante du problème de placement de pôles n'est pas assurée. En fait il faut comprendre que si un algorithme numérique est composé de plusieurs sous-algorithmes ou sous-problèmes, et que si l'un de ces sous-problèmes est mal conditionné ou mal posé, alors la fiabilité du résultat final est compromise. Pour notre exemple, nous avons deux sous-problèmes, d'abord la transformation d'état qui emmène à la construction de la matrice T, et ensuite la solution du système $FT = \bar{F}$. Pour construire une matrice de transformation d'espace d'état mieux conditionnée et ainsi avoir plus de chances d'obtenir un bon résultat, d'autres techniques d'algèbre linéaire numérique sont utilisées, en particulier les transformations orthogonales [17].

Transformations orthogonales

Une matrice $Q \in \mathbb{R}^{n\times n}$ est orthogonale (*orthogonal matrix*) si $Q^TQ = QQ^T = I$. Ainsi, toutes les valeurs singulières de Q sont égales à 1. La matrice Q est donc bien conditionnée:

$$\kappa_2(Q) = 1, \qquad \kappa_*(Q) < \kappa_\infty(Q) < n.$$

Une autre propriété importante des matrices orthogonales est qu'elles conservent par multiplication le conditionnement κ_2 d'une matrice quelconque:

$$\kappa_2(AQ) = \kappa_2(QA) = \kappa_2(A).$$

Supposons qu'on applique une transformation d'état orthogonale Q telle que

$$\bar{A} = QAQ^T = \begin{bmatrix} -a_{n-1} & -a_{n-1} & \cdots & -a_0 \\ \alpha_1 & \times & & \\ & \ddots & \ddots & \\ & & \alpha_{n-1} & \times \end{bmatrix}, \qquad \bar{B} = QB = \begin{bmatrix} b_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{2.9}$$

où $\times$ représente un élément quelconque possiblement différent de zéro. Supposons aussi que $\bar{F}$ soit choisie telle que la boucle fermée ait le polynôme caractéristique désiré

$$\det(sI - \bar{A} - \bar{B}\bar{F}) = f(s).$$

Alors, la solution au problème de placement de pôles est simplement $F = \bar{F}Q$. Reste à déterminer $\bar{F}$. À partir de la définition des valeurs et vecteurs propres, on peut vérifier que

$$\bar{A} + \bar{B}\bar{F} = V\Lambda V^{-1}$$

avec

$$\Lambda = \text{diag}\{\lambda_1, \ldots, \lambda_n\}, \qquad V = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix}$$

où λ_i, $i = 1{:}n$ sont les valeurs propres de $\bar{A} + \bar{B}\bar{F}$ (racines de $f(s)$) et v_i, $i = 1{:}n$ sont les vecteurs propres correspondants. Notons qu'on peut construire les vecteurs v_i à partir des $n-1$ dernières lignes de $\bar{A} + \bar{B}\bar{F}$

$$\begin{bmatrix} \alpha_1 & \times - \lambda_i & & \\ & \ddots & \ddots & \\ & & \alpha_{n-1} & \times - \lambda_i \end{bmatrix} v_i = \Gamma_i v_i = 0, \tag{2.10}$$

et donc, extraire facilement $\bar{F}$ en divisant la première ligne de $\bar{A} - V\Lambda V^{-1}$ par b_1.

La solution de (2.10) peut être obtenue aussi grâce à des transformations orthogonales et ainsi assurer de bonnes propriétés numériques. En particulier on peut appliquer sur Γ_i une factorisation LQ, duale de la factorisation QR [33] (voir §2.3.1 de cette thèse) pour obtenir une base de son espace nul.

Le système transformé (2.9) est nommé la forme commandable de Hessenberg (à comparer avec la forme commandable (2.4)). La matrice orthogonale Q n'est qu'une composition de matrices de Householder (*Householder matrix*) [33] (voir §2.3.1 de cette thèse) appliquées pour annuler les éléments désirés dans A et B [17].

Cette méthode pour le placement de pôles est présentée dans [70, 71]. Les idées dans ces articles sont à la base de plusieurs autres algorithmes, en particulier de ceux présentés dans [53] sur lesquels est basée la fonction `place` de la Control System Toolbox.

Commandabilité du système

La matrice Q en (2.9) est après tout une transformation de similarité pour l'état du système. Alors, les propriétés du système original (A, B) sont aussi celles du système transformé $(\bar{A}, \bar{B})$. En particulier, notons que

$$\bar{\mathcal{C}} = \begin{bmatrix} QB & QAQ^TQB & \cdots & (QAQ^T)^{n-1}QB \end{bmatrix} = Q\mathcal{C}. \tag{2.11}$$

La commandabilité du système reste invariante car le rang de $\bar{\mathcal{C}}$ est égal au rang de $\mathcal{C}$. La différence est que l'algorithme orthogonal pour le placement pôles ne dépend pas de l'inversibilité de $\mathcal{C}$. Une fois que $\bar{F}$ est déterminée, on peut toujours calculer F. Alors, où intervient la condition que le système doit être commandable? La réponse se trouve, bien sûr, dans la partie où la plupart du calcul numérique est effectué, c'est-à-dire, dans la solution du système (2.10) pour les n valeurs propres désirées. Notons que si Γ_i n'a pas de rang plein par lignes, alors le calcul de son espace nul devient un problème mal posé car une petite perturbation des éléments de Γ_i peut changer la dimension de l'espace nul. Alors, on requiert que Γ_i ait rang plein, autrement dit, que α_i, $i = 1{:}n-1$ soient différents de zéro. Maintenant, si on observe soigneusement

la matrice de commandabilité

$$\bar{\mathcal{C}} = \begin{bmatrix} b_1 & \times & \times & \times & \cdots & \times \\ & b_1\alpha_1 & \times & & & \times \\ & & b_1\alpha_1\alpha_2 & \times & & \times \\ & & & b_1\alpha_1\alpha_2\alpha_3 & & \vdots \\ & & & & \ddots & \times \\ & & & & & b_1\alpha_1\cdots\alpha_{n-1} \end{bmatrix} \tag{2.12}$$

on peut vérifier que la condition $\alpha_i \neq 0$, $i = 1{:}n-1$ implique que le système est commandable, $\text{rank}(\mathcal{C}) = n$.

Finalement, à partir de (2.11) et (2.12) notons que $Q^T\bar{\mathcal{C}} = \mathcal{C}$. Alors, Q peut être obtenue à partir de la factorisation QR de $\mathcal{C}$. Pour des raisons numériques, on construit Q directement à partir de A et B. Cependant, il est important de noter que la factorisation QR peut nous donner une approximation plus utile du rang de $\mathcal{C}$ (ou de la commandabilité du système) que celle obtenue à partir des valeurs singulières.

Exemple 2.3
Considérons la matrice de commandabilité $\mathcal{C}$ de l'exemple (2.2). En obtenant sa factorisation QR, on observe les valeurs suivantes sur la diagonale du facteur triangulaire $\bar{\mathcal{C}}$:

-1.476..., 3.417..., -1.836..., 1.447..., -0.311..., -0.142..., 0.008...

Ainsi le système est commandable et on peut appliquer l'algorithme décrit ci-dessus pour le placement de ses pôles. Le système est commandable, mais ça n'empêche pas que $\mathcal{C}$ soit très mal conditionnée. Autrement dit, si par contre on veut placer les pôles avec la formule d'Ackermann, on n'obtiendra pas de résultats satisfaisants. ■

En conclusion, le calcul du rang d'une matrice est un exercice d'interprétation des résultats numériques, et cette interprétation ainsi que les méthodes numériques utilisées dépendent du contexte. Si on compte inverser la matrice analysée alors il vaut mieux estimer son rang à partir de ses valeurs singulières car elles sont directement liées au conditionnement et à la distance à la singularité de la matrice.

2.2 Approche polynomiale

Comme brièvement rappelé dans la section antérieure, le succès des logiciels de CACSD par approche espace d'état est fortement basé sur la théorie bien consolidée de l'algèbre linéaire numérique et les logiciels de calcul numérique existants. Cependant, notons que cette théorie n'est pas propre à l'approche espace d'état. C'est le fait que le système soit représenté par des matrices et vecteurs constants qui nous permet de l'appliquer. Par contraste, avec l'approche polynomiale[7] on représente le

7. On regroupe ici l'approche par équations polynomiales [58] et l'approche comportementale [83].

système sous la forme (2.3). Les objets mathématiques du modèle sont alors les polynômes et les matrices polynomiales, et malheureusement il n'existe pas encore de théorie d'analyse numérique solide pour ces objets.

Le développement de logiciels de CACSD pour les polynômes est très restreint. À notre connaissance, parmi les logiciels existants qui permettent la manipulation des polynômes et matrices polynomiales[8] seulement la Polynomial Toolbox pour Matlab[9] [84] est orientée vers les systèmes en commande et n'exécute quasiment que des calculs numériques, et non symboliques[10].

Quelques autres exemples de logiciels numériques de CACSD pour l'approche polynomiale sont issus des efforts de quelques groupes de chercheurs et étudiants[11] mais leur diffusion est encore très limitée et il manque une analyse rigoureuse des propriétés numériques. Comme pour l'approche espace d'état, mais à une petite échelle, la variété des interfaces, des méthodes et des types d'implémentation rend très difficile la definition d'un logiciel standard pour l'approche polynomiale. Cependant, les mêmes caractéristiques générales énoncées dans la section 2.1 peuvent s'appliquer aux logiciels de CACSD pour l'approche polynomiale.

Par rapport à l'interface pour l'utilisateur, la conception d'un logiciel de CACSD pour les méthodes polynomiales implique de mettre en place une structure qui permet à l'ordinateur d'accepter, de stocker et de représenter les variables (indeterminées) des polynômes[12]. Il est clair que cette interface peut nécessiter des temps de calcul importants dans certains cas.

Exemple 2.4
Prenons les cas de la multiplication $C(s) = A(s)B(s)$ avec $A(s) = A_0 + A_1 s + \cdots \in \mathbb{R}^{15\times 15}[s]$ et $B(s) = B_0 + B_1 s + \cdots \in \mathbb{R}^{15\times 15}[s]$ deux matrices polynomiales de degrés 10 et 12 respectivement. En général $C(s) = C_0 + C_1 s + \cdots \in \mathbb{R}^{n\times n}[s]$ a degré 22 et peut s'obtenir numériquement avec l'opération équivalente

$$\begin{bmatrix} C_0 & \cdots & C_{22} \end{bmatrix} = \begin{bmatrix} A_0 & \cdots & A_{10} \end{bmatrix} \begin{bmatrix} B_0 & \cdots & B_{12} & & \\ & \ddots & & \ddots & \\ & & B_0 & \cdots & B_{12} \end{bmatrix}$$

sur les matrices réelles. Avec une machine SUN Microsystems Ultra 5 (SunOS 5.9) et Matlab 7, la version 3 de la Polynomial Toolbox a besoin de 16.3 secondes pour

8. Par exemple: Maple, Mathematica, MACSYMA, Reduce, MuMath, etc...

9. Voir `www.polyx.com`.

10. Comme on a mentionné dans le Chapitre 1, le calcul symbolique n'est pas toujours adéquat pour résoudre des problèmes avec des coefficient à précision finie. Il vaut mieux (au niveau précision ainsi que temps de calcul) transformer le problème polynomial en un problème constant équivalent sur lequel on peut appliquer des méthodes numériques, cf. la section 2.3 de cette thèse. Malheureusement, tous les problèmes en commande avec des polynômes ou des matrices polynomiales ne sont pas numériquement réductibles (*numerically reducible*). Par conséquent, l'application et l'analyse des algorithmes symboliques reste importante pour certaines méthodes polynomiales en commande [76].

11. Voir `http://ar.c-a-k.cz/polynomial/software.html`.

12. Noter cependant qu'aucun calcul symbolique n'est considéré, on se restreint aux logiciels numériques.

exécuter l'instruction `mtimes(A,B)` et dévoiler le résultat, alors que le temps d'exécution de la multiplication matricielle réelle ci-dessus est seulement de 0.3 secondes. Les 16 secondes de différence sont utilisées uniquement pour l'interface (majoritairement pour représenter $C(s)$ sous la forme d'une matrice avec ses éléments qui sont des polynômes). ■

Evidemment, une programmation soignée peut améliorer l'interface. On peut même la réduire au minimum (si l'utilisateur est initié) et présenter les matrices polynomiales comme un ensemble de coefficients.

La plupart des logiciels de CACSD pour l'approche polynomiale utilisent ou peuvent utiliser les bibliothèques numériques de base comme LAPACK ou BLAS. Malheureusement la capacité de ces bibliothèques numériques n'est pas toujours exploitée au maximum. Pour illustrer ceci prenons le cas de la multiplication de l'Exemple (2.4). LAPACK (BLAS) est appelé à travers Matlab quand la Polynomial Toolbox exécute l'instruction `mtimes(a,b)`. L'opération exécutée par l'ordinateur est une opération du niveau 3 de BLAS[13]. Notons, néanmoins, qu'en utilisant la capacité de LAPACK (BLAS) pour manipuler les matrices avec un motif de zéros défini (matrices creuses, *sparse matrices*) et avec une structure particulière (matrices structurées, *structured matrices*), le calcul de $C(s)$ pourrait être plus efficace. Ainsi, une meilleure exploitation des bibliothèques ne peut être assurée que par l'intermédiaire d'une meilleure mis en ouvre (programmation).

2.2.1 Problèmes numériques et polynômes

Quel est le vrai problème des logiciels de CACSD pour l'approche polynomiale? On essaiera de répondre à cette question dans les paragraphes suivants. Dans la littérature technique sur les problèmes numériques en commande, deux critiques sont généralement faites aux méthodes polynomiales :

1. Avec l'approche polynomiale, les problèmes d'analyse et de synthèse en commande des systèmes linéaires sont généralement réduits à l'analyse des propriétés des matrices polynomiales en tant qu'objets algébriques (obtention de la structure propre, le rang, etc.), la manipulation de ces matrices polynomiales (factorisation, triangularisation, etc.) et la solution d'équations polynomiales ou équations Diophantiennes [59]. Traditionnellement les méthodes pour accomplir ces tâches sont basées sur les opérations élémentaires dans l'anneau des polynômes (*elementary polynomial operations*) et il est prouvé que ces opérations peuvent causer une instabilité numérique [27, 108].
2. Bien qu'au niveau théorique les représentations (2.2) et (2.3) sont équivalentes, au niveau numérique la réputation des polynômes comme objets de modélisation est plutôt mauvaise. Les polynômes sont vus, en général, comme des objets qui génèrent des problèmes mal conditionnés (voir mal posés) [45].

13. Une opération du niveau 3 signifie une opération matrice–matrice.

L'instabilité est spécifique à l'algorithme utilisé. Dans ce sens là, l'instabilité est due à l'application des opérations élémentaires et non pas à l'approche polynomiale elle-même. On peut, donc, en utilisant d'autres types d'opérations, concevoir des algorithmes numériques stables. Malheureusement, la stabilité d'un algorithme n'assure pas l'exactitude de la solution calculée, cette propriété dépend aussi du conditionnement du problème, cf. Exemple 2.1. Le conditionnement des problèmes polynomiaux qu'on rencontre en commande (via l'approche polynomiale) est en général élevé. Cependant, comme on le verra par la suite, de nouveaux résultats laissent supposer qu'on peut contourner ces inconvénients.

L'instabilité potentielle des opérations élémentaires sur l'anneau des polynômes est concentrée dans le choix d'un *pivot*[14]. Le choix du pivot polynomial se base essentiellement sur son degré, et ne pas considérer ses coefficients peut introduire des erreurs numériques importantes.

Exemple 2.5
On veut obtenir la forme de Smith $S(s)$ de la matrice polynomiale

$$F(s) = \begin{bmatrix} s & 0 & 1 & \alpha \\ 0 & s & 1 & 1 \\ 1 & 0 & s+1 & 0 \\ 0 & 1 & 1 & s+1 \end{bmatrix}.$$

Après quelques opérations polynomiales élémentaires par lignes et par colonnes groupées dans $L_1(s)$ et $C_1(s)$ respectivement on obtient

$$L_1(s)F(s)C_1(s) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & -s & 0 \\ 0 & 1 & 0 & -s \end{bmatrix} F(s) \begin{bmatrix} 1 & 0 & -s-1 & 0 \\ 0 & 1 & -1 & -s-1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -s^2-s+1 & \alpha \\ 0 & 0 & 1-s & -s^2-2+1 \end{bmatrix} = S_1(s).$$

Jusqu'ici nous n'avons pas de problèmes numériques. Pour continuer jusqu'à l'obtention de la forme de Smith, on choisit comme pivot suivant l'élément de plus petit degré dans la sous-matrice $S_1(3{:}4, 3{:}4)$, c'est-à-dire α. Avec ce pivot on introduit des zéros dans les positions correspondantes. Ainsi on obtient

$$L(s)F(s)C(s) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & 0 \\ 0 & 0 & 0 & -\gamma s^4 - 2\gamma s^3 + \gamma s^2 + (2\gamma - 1)s + (1-\gamma) \end{bmatrix} = S(s)$$

14. Un élément sur une ligne ou colonne à partir duquel on peut introduire des zéros sur une matrice ou vecteur à l'aide d'opérations par lignes ou colonnes.

avec $\gamma = 1/\alpha$ et

$$L(s) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & -s & 0 \\ \gamma s^2 + \gamma s - \gamma & 1 & -\gamma s^3 - \gamma s^2 + \gamma s & -s \end{bmatrix},$$

$$C(s) = \begin{bmatrix} 1 & 0 & 0 & -s-1 \\ 0 & 1 & -s-1 & -\gamma s^3 - 2\gamma s^2 + \gamma - 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & \gamma s^2 + \gamma s - \gamma \end{bmatrix}.$$

Si α est très petit, alors γ devient très grand. Dans ce cas notons que, suivant la précision utilisée, après l'arrondi inévitable de l'ordinateur, les matrices calculées deviennent $\hat{L}(s) = L(s)$, $\hat{C}(s) = C(s)$ et

$$\hat{S}(s) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & 0 \\ 0 & 0 & 0 & -\gamma s^4 - 2\gamma s^3 + \gamma s^2 + 2\gamma s - \gamma \end{bmatrix}$$

ce qui démontre l'instabilité de l'algorithme. En fait la matrice $\hat{S}(s)$ n'est pas la forme de Smith de $A(s)$ légèrement perturbée,

$$\hat{L}^{-1}(s)\hat{S}(s)\hat{C}^{-1}(s) = \begin{bmatrix} s & 0 & 1 & \alpha \\ 0 & s & s & 1 \\ 1 & 0 & s+1 & 0 \\ 0 & 1 & 1 & s+1 \end{bmatrix} \neq F(s),$$

car l'information sur l'élément $(2,3)$ de $F(s)$ s'est complètement perdue. ∎

Les opérations polynomiales élémentaires ont leur intérêt et leur importance car elles constituent l'outil mathématique avec lequel s'explique la théorie de l'approche polynomiale. Cependant, il est clair que malgré cet intérêt académique, les algorithmes polynomiaux classiques doivent s'adapter pour prendre en compte les problèmes numériques. Ce besoin d'adaptation a aussi été observé dans l'approche espace d'état et l'algèbre linéaire numerique, cf. la section 2.1. Notons que le phénomène illustré dans l'exemple antérieur ressemble à ce que l'on observe avec l'élimination Gaussienne [33], où choisir un pivot uniquement par sa position sur la diagonale peut causer les mêmes problèmes numériques[15].

La solution donnée à ces problèmes par l'algèbre linéaire numérique se dénomme stratégie de pivot (*pivoting*) [33]. Il y a plusieurs stratégies de pivot mais l'idée générale consiste à choisir comme pivot l'élément de plus grande magnitude. Malheureusement ces stratégies de pivot ne sont pas toujours applicables dans le cas des opérations polynomiales élémentaires, comme pour l'Exemple 2.5.

15. Considérer par exemple la matrice $\begin{bmatrix} \alpha & 1 \\ 1 & 1 \end{bmatrix}$ avec α très petit.

Pour contourner les problèmes de stabilité on doit alors éviter les opérations élémentaires en ne manipulant que les coefficients de la matrice polynomiale analysée, à l'aide des méthodes fiables de l'algèbre linéaire numérique. Pour l'Exemple 2.5 la solution est relativement facile. Notons tout d'abord que $A(s)$ est un faisceau et donc que ses zéros finis peuvent être obtenus par la factorization QZ, qui est une méthode backward stable [33, 72]. Dans notre exemple cela équivaut à trouver les valeurs propres de A_0. Ensuite, à l'aide de la forme canonique de Kronecker [105], on détermine la multiplicité algébrique et géométrique de chaque zéro, et avec cette information on construit les polynômes invariants de $A(s)$, c'est-à-dire les éléments sur la diagonale de sa forme de Smith.

Autres méthodes évitant les opérations polynomiales élémentaires seront l'objet des prochains chapitres de cette thèse.

Les mauvaises propriétés numériques des polynômes, en tant qu'objets de modélisation, sont liées principalement à la surcondensation des donneés. L'exemple suivant extrait de [45] est très illustratif.

Exemple 2.6
On veut calculer les valeurs propres de la matrice

$$A = Q^T \text{diag}\{1, 2, \ldots, 20\} Q$$

avec Q une matrice orthogonale quelconque. La matrice A est symétrique et donc par la factorisation QR symétrique (qui est stable [33]) on peut obtenir les valeurs propres de A avec une très bonne précision. Notons aussi que A est parfaitement bien conditionnée pour le problème des valeurs propres car ses valeurs propres coïncident avec ses valeurs singulières [81]. Par contre, si on essaye de calculer les valeurs propres de A à partir de la définition classique, c'est-à-dire comme les racines du polynôme caractéristique $p(s) = \det(sI - A)$, les résultats obtenus seront très inexacts. Par exemple, avec la fonction `roots` de Matlab, les 3 plus grandes racines de $p(s)$ calculées sont: 20.0003, 18.9970 et 18.0117. Notons que les coefficients du polynôme caractéristique varient entre 1 et 2.43×10^{18} et donc on aura des problèmes pour les représenter avec une précision finie. ∎

La surcondensation des données est évidente: au lieu de considérer les 210 coefficients de la matrice A, on ne considère que 20 coefficients du polynôme $p(s)$. Malheureusement ces coefficients varient dans une amplitude que l'ordinateur ne peut pas manipuler avec précision. D'autre part, il faut noter que ce problème n'est pas propre aux polynômes, dans l'Exemple 2.2 on a vu comme les normes des colonnes d'une matrice peuvent varier aussi avec une amplitude très vaste[16].

Il est certain que la formulation du problème en termes polynomiaux n'est pas la meilleure méthode pour cet exemple. Cependant, se baser sur des exemples comme celui-ci pour en conclure qu'il vaut mieux éviter les polynômes, nous prive d'une

16. On peut penser aussi, comme pour les matrices constantes, à appliquer un scaling pour équilibrer les magnitudes des coefficients d'un polynôme. On mentionne cette possibilité dans la section 3.1.

alternative qui peut être tout aussi efficace qu'une autre, voire parfois meilleure. On peut toujours trouver des exemples pour lesquels les méthodes polynomiales ont une meilleure performance. Evidemment, pour chaque exemple il existera sûrement un contrexemple, raison pour laquelle nous ne poursuivrons pas cette discussion que nous croyons sans intérêt.

Il est vraiment important de noter que, tout comme au niveau théorique chaque approche a ses avantages et désavantages et surtout sa propre procédure d'analyse, au niveau numérique on doit résoudre des problèmes différents qui doivent s'analyser de façon différente.

Reprenons l'Exemple 2.6, pour lequel on a décrit deux problèmes différents dont les solutions, avec une précision infinie, sont équivalentes. D'un côté on a le problème de l'obtention des valeurs propres pour lequel la matrice A est très bien conditionnée, et de l'autre côté le problème du calcul des racines du polynôme caractéristique. C'est en analysant ce dernier problème qu'on conclut que le polynôme $p(s)$ est mal conditionné pour le problème de l'obtention de ses racines. Par contre, si on compare les racines de $p(s)$ avec les valeurs propres de A, la seule chose qu'on peut conclure c'est que la méthode qui consiste à obtenir les valeurs propres d'une matrice en calculant les racines du polynôme caractéristique n'est pas stable.

En conclusion si on a décidé d'appliquer l'approche polynomiale ou s'il on a un problème représenté avec des polynômes, l'analyse doit être faite dans le cadre des polynômes. Cela dit, il est important de consolider une théorie d'algèbre polynomiale numérique qui adapte l'analyse numérique aux polynômes. On a vu que cette théorie est actuellement en train de se développer sur les bases de la géométrie algébrique et de l'algèbre commutative [98].

Nous croyons donc qu'en contournant les opérations polynomiales élémentaires, et avec une algèbre polynomiale numérique comme outil d'analyse, il est uniquement une question de temps pour que le développement de logiciels de CACSD pour l'approche polynomiale rattrape celui de l'approche espace d'état.

2.3 Méthodes pour les matrices polynomiales

Il existe différentes approches pour résoudre numériquement les problèmes en commande par approche polynomiale. Ici nous les regroupons en deux grandes catégories[17]:

– Exprimer le problème polynomial comme un problème de commande d'un système en espace d'état et appliquer les méthodes numériques et logiciels de CACSD propres à cette approche, cf. la section 2.1 de cette thèse. Ensuite la solution au problème en espace état est reformulée en termes du problème

17. Voir [37] pour une classification plus détaillée. Ici on parle uniquement des problemes polynomiaux qui sont numériquement réductibles. Cependant ils existent des problemes, qui n'ont pas cette propriété, pour lesquels le calcul symbolique est appliqué mais on ne les étude pas dans cette thèse, voir [76].

polynomial original. Un exemple clair de cette procédure est représenté par les algorithmes pour la factorisation spectrale des matrices polynomiales via la solution des équations de Riccati [95], cf. Chapitre 5 de cette thèse et [1]. L'obtention du descripteur du système est aussi une procédure qui permet, via l'espace d'état, de résoudre des problèmes avec les matrices polynomiales [110, 111], cf. Chapitre 4 de cette thèse. Voir aussi [54, 55] pour d'autres algorithmes d'espace d'état pour les matrices polynomiales.

– Une façon plus directe de travailler consiste en manipuler directement les coefficients des matrices polynomiales pour résoudre un problème constant équivalent mais sans une reformulation en espace état. La linéarisation d'une matrice polynomiale via la construction de sa forme compagne est une procédure très utilisée. Noter que la forme compagne résultante de la linéarisation est un faisceau sur lequel on peut appliquer des algorithmes de l'algèbre linéaire numérique [107], cf. Chapitres 3 et 4 de cette thèse. Voir également le point 4.3 du rapport prospectif [20], ou encore [49]. Cependant, strictement parlant, la linéarisation est aussi une réalisation d'état de la matrice linearisée. La construction de matrices de Sylvester[18] (*Sylvester matrices*) avec les coefficients des matrices polynomiales [94], par contre, est une procédure encore plus directe car le problème polynomial original est directement formulé via ses coefficients, cf. Chapitres 3, 4 et 5. L'interpolation et l'utilisation de la transformée rapide de Fourier sont aussi des méthodes prometteuses qui ne passent pas par l'espace état [47].

Dans cette thèse on s'intéresse au développement d'algorithmes numériques suivant la deuxième approche. La structure générale de ces algorithmes est la suivante:

Algorithme 2.1 *(Algorithme général)*

1. *Déterminer un problème constant équivalent (*constant equivalent problem*, CEP) au problème polynomial original (*original polynomial problem*, OPP) sur lequel on peut appliquer des méthodes numériques;*
2. *Résoudre le CEP avec des algorithmes fiables de l'algèbre linéaire numérique;*
3. *Obtenir la solution du OPP à partir de la solution calculée du CEP.*

Exemple 2.7
Considérons le problème de l'obtention des zéros finis[19] de la matrice polynomiale

$$A(s) = A_2 s^2 + A_1 s + A_0 = \begin{bmatrix} -1 - s + s^2 & -\alpha \\ -1 - s & -1 - s + s^2 \end{bmatrix}.$$

On peut démontrer que les valeurs propres généralisées du faisceau[20] sous la forme compagne

$$F(s) = sF_1 + F_0 = s \begin{bmatrix} I & 0 \\ 0 & A_2 \end{bmatrix} + \begin{bmatrix} 0 & -A_0 \\ I & -A_1 \end{bmatrix}$$

18. Matrices structurées (Toeplitz ou Toeplitz par blocs) et constantes.
19. Les zéros finis ou valeurs propres de $A(s)$ sont les valeurs de s pour lesquelles $A(s)$ perd son rang, cf. Chapitre 3 de cette thèse.
20. Matrice polynomiale de premier degré.

sont égales aux zéros recherchés. Le premier pas de l'Algorithme 2.1 est la construction de $F(s)$. Ensuite les pas 2 et 3 consistent à obtenir la factorisation QZ de $F(s)$

$$\bar{F}_1 = QF_1Z, \qquad \bar{F}_0 = QF_0Z$$

et à evaluer le quotient des valeurs diagonales de $\bar{F}_0$ et $\bar{F}_1$ [72]. Notons que pour cet exemple, la forme compagne de $A(s)$ résulte en la matrice $F(s)$ de l'Exemple 2.5. Donc, la solution à notre OPP est simplement l'ensemble des valeurs propres de F_0. ■

La complexité algorithmique[21] (*algorithmic complexity*) dépend du nombre d'opérations arithmétiques élémentaires en virgule flottante (*floating point operations, flops*) comme la multiplication ou la somme, effectuées à chaque pas. Par conséquent, on souhaite que le CEP admette une formulation simple et également que sa solution soit facilement traduite en la solution du OPP. L'idée est donc de concentrer le gros du calcul sur la solution du CEP. Cet objectif n'est pas toujours atteint. Ainsi dans le Chapitre 4 nous décrirons des algorithmes où le calcul réalisé pour les pas 1 et 3 de l'Algorithme (2.1) peut être plus coûteux que celui requis pour le pas 2.

Pour analyser la stabilité de l'Algorithme 2.1 on le divise en 3 sous-algorithmes. Il est facile de montrer que si l'un des sous-algorithmes est instable, alors l'algorithme global est instable [43, 45]. On s'intéresse donc à la stabilité de chaque sous-algorithme. En particulier, il est désirable que l'algorithme de solution du CEP (pas 2) soit stable. Malheureusement cet objectif ne garantit pas la stabilité de l'algorithme global. Comme rappelé dans la section 2.2.1., l'analyse doit être faite à partir des polynômes. Par conséquent, non seulement la solution du CEP, mais aussi l'erreur arrière de la méthode numérique appliquée au CEP doivent être traduites en termes des coefficients des polynômes ou des matrices polynomiales analysées. Parfois on verra que de petites erreurs dans les coefficients du CEP sont traduites en des erreurs considérables dans les coefficients du OPP [101], cf. la section 3.1 dans cette thèse.

Il est possible d'obtenir une borne supérieure de l'erreur arrière pour le OPP à partir de l'erreur arrière pour le CEP. Dans cette thèse (Chapitre 4) on fait ce type d'analyse. Cependant, on peut vérifier que cette borne supérieure est parfois pessimiste. C'est l'analyse de la géométrie du problème et l'algèbre polynomiale numérique qui permettront d'obtenir des bornes plus précises, ainsi que de comprendre la sensibilité (conditionnement) des polynômes ou des matrices polynomiales analysées. Cette étude dépasse les objectifs de cette thèse. Néanmoins on consacre quelques lignes dans la section 3.1 pour illustrer ces idées, en analysant quelques algorithmes existants pour le problème des valeurs propres d'une matrice polynomiale. L'application de ces techniques aux nouveaux algorithmes présentés dans cette thèse est proposée comme travail futur.

On a vu dans l'Exemple 2.7 que la linéarisation d'une matrice polynomiale est une procédure qui entre dans le cadre du pas 1 de l'Algorithme 2.1. Le CEP qu'on analyse

21. Estimation du nombre d'opérations effectuées par l'ordinateur en exécutant un algorithme. Cette estimation est représentée comme $O(n^N)$: de l'ordre de n^N où n est un paramètre du problème, comme la dimension, et N est un entier positif [33].

ensuite se pose sur le faisceau obtenu et peut être résolu en appliquant des méthodes numériques comme l'obtention d'une forme canonique de Kronecker, ou une factorisation QZ par exemple [105]. Cette approche appelée par la suite l'approche des faisceaux (*pencil approach*) est très répandue dans la littérature technique, en particulier pour l'obtention de la structure propre d'une matrice polynomiale, problème que l'on analyse dans les Chapitres 3 et 4 de cette thèse. D'autre part, le CEP peut être formulé aussi à l'aide des matrices de Sylvester. Les matrices de Sylvester sont dérivées naturellement des matrices polynomiales. Elles s'obtiennent en arrangeant les coefficients de la matrice polynomiale dans une matrice Toeplitz par blocs, cf. l'Exemple 2.4. Dans les Chapitres 3 et 4 on montre que cette approche, appelée par la suite l'approche Toeplitz (*Toeplitz approach*) est très efficace pour le problème du calcul de la structure propre d'une matrice polynomiale, et que c'est une alternative fiable à l'approche des faisceaux.

Les opérations qu'on effectue sur les matrices Toeplitz sont en général des factorisations pour obtenir les rangs, les espaces nuls ou pour résoudre des systèmes d'équations. Pour accomplir ces tâches plusieurs méthodes numériques peuvent être utilisées: la décomposition en valeurs singulières, l'élimination Gaussienne avec pivot, les factorisations orthogonales comme la factorisation QR, ou encore d'autres méthodes qui servent a dévoiler le rang d'une matrice (*rank revealing methods*, RRM) [15]. Toutes ces méthodes sont très bien étudiées dans la littérature [33] et leur bonnes propriétés numériques sont prouvées [43, 99]. Ici on décrit brièvement quelques propriétés des méthodes qu'on utilise dans cette thèse.

2.3.1 Décomposition en valeurs singulières

La décomposition en valeurs singulières (*singular value décomposition*, SVD)

$$\Sigma = P^T M Q$$

d'une matrice constante $M \in \mathbb{R}^{m\times n}$ peut être implémentée comme un algorithme numériquement stable[22] [33]. Dans ce cas, si $\hat{P}$, $\hat{Q}$ et $\hat{\Sigma}$ sont les matrices calculées, on peut démontrer [99] que

$$\hat{\Sigma} = \hat{P}^T(M+\Delta)\hat{Q}$$

avec

$$\|\Delta\|_2 \leq O(\varepsilon)\|M\|_2.$$

De plus, si $\hat{\sigma}_i$, $i = 1{:}\min(m,n)$ sont les valeurs singulières calculées de M, alors

$$|\sigma_i - \hat{\sigma}_i| \leq O(\varepsilon)\|M\|_2.$$

Alors, on peut déterminer le rang de M, dénoté $\rho = \mathrm{rank}(M)$, en appliquant la SVD et en comptant le nombre de valeurs singulières inférieures à $\xi\|M\|_2$, où ξ est un seuil dépendant de ε, cf. l'Exemple 2.2. Il en découle que $\|M\hat{Q}(:,\rho+1:n)\|_2$ est

22. L'implémentation fiable de la SVD n'est pas facile. Il existe différentes implémentations, mais cela dépasse le cadre de cette thèse. Pour plus de détails voir par exemple [33].

suffisamment faible et que la matrice $\hat{Q}(:, \rho+1:n)$ constitue une base calculée de l'espace nul de M.

La SVD est une procédure très efficace pour déterminer le rang d'une matrice, cependant elle a deux inconvénients. D'une part, elle est relativement lourde à calculer, nécessitant $4m^2n + 8mn^2 + 9n^3$ flops [33]. D'autre part, elle ne permet pas facilement la mis à jour (updating): si une ligne ou colonne est rajoutée à M, les matrices P et Q ne sont pas mises à jour facilement.

2.3.2 Factorisation LQ

Une autre RRM plus facile à exécuter que la SVD est la factorisation LQ. La factorisation

$$M = LQ^T$$

de la matrice M est la duale de la factorisation QR définie dans [33]. La matrice L est sous forme triangulaire inférieure, et la matrice Q est orthogonale. Le nombre de flops requis pour la factorisation LQ est approximativement $4(m^2n - mn^2 + \frac{1}{3}n^3)$ [33].

Pour que la factorisation LQ puisse révéler le rang de M, une stratégie de pivot est nécessaire. En pratique, au moment de calculer la factorisation, on applique au pas j une matrice de Householder pour annuler les $n-j$ éléments de droite d'une ligne préalablement choisie. Pour choisir cette ligne, plusieurs stratégies peuvent être suivies. Par exemple, on peut calculer $\|M(i, j+1:n)\|$ pour toutes les lignes indexées par i qui n'ont pas encore un pivot, et choisir celle qui a la plus grande norme. Notons que le pivotage permet de connaître la position des lignes de M qui sont linéairement indépendantes. Alors, une simple permutation par lignes, représentée par la matrice P, peut être appliquée à L pour la mettre sous la forme

$$PL = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \tag{2.13}$$

où L_{11} est de dimension $\rho \times \rho$ avec des éléments différents de zéro (les pivots) sur sa diagonale.

Tout comme la SVD, la factorisation LQ est une méthode numériquement stable. On peut démontrer [43, 99] que les matrices calculées $\hat{L}$ et $\hat{Q}$ satisfont

$$(M + \Delta)\hat{Q} = \hat{L}$$

avec

$$\|\Delta\|_2 \leq O(\varepsilon)\|M\|_2.$$

De plus, si on partitionne L comme en (2.13), alors

$$\frac{\|\hat{L}_{22} - L_{22}\|_2}{\|M\|_2} \leq \frac{\|\Delta\|_2}{\|M\|_2}(1 + \|L_{11}^{-1}L_{21}^T\|_2) \tag{2.14}$$

où $\|L_{11}^{-1}L_{21}^T\|_2$, le conditionnement de la factorisation LQ pour les matrices de rang déficient, est généralement faible [42]. Donc, à partir de l'equation (2.14) on peut

déterminer le rang de M en appliquant la factorisation LQ et en comptant le nombre de colonnes de $\hat{L}_{22}$ telles que $\|\hat{L}_{22}\|_2 \leq \xi\|M\|_2$ avec ξ un seuil qui dépend de ε. Si $\text{rank}(M) = \rho$, alors $\|M\hat{Q}(:, \rho+1 : n)\|_2$ est suffisamment faible et la matrice $\hat{Q}(:, \rho+1 : n)$ constitue une base calculée de l'espace nul de M.

L'inconvénient de la factorisation LQ en tant que RRM est que la norme $\|\hat{L}_{22}\|_2$ ne diminue pas toujours suffisamment rapidement. Heureusement, ce problème se présente seulement pour des matrices soigneusement choisies, cf. l'Exemple 5.5.1 dans [33].

2.3.3 Forme Echelon L

Supposons qu'on veuille réduire encore plus la complexité algorithmique par rapport aux méthodes SVD et LQ des sections précédentes. Une possibilité est de calculer la factorisation LQ sans mettre à jour et stocker la matrice Q, qui est une succession de matrices de Householder H_i appliquées à la matrice M. Le calcul de la matrice L uniquement requiert $2(mn^2 - \frac{1}{3}n^3)$ flops [33]. Noter qu'avec cette méthode, appelée *forme échelon L*, on ne peut pas récupérer une base orthogonale de l'espace nul à droite de M. Cependant, à partir de L on peut construire une base naturelle (non orthogonale en général) de l'espace nul à gauche en résolvant quelques systèmes d'équations triangulaires. Par exemple supposons que pour une matrice M donnée, après l'application de 3 matrices de Householder, on obtienne la forme échelon

$$L = MH_1H_2H_3 \begin{bmatrix} \times_{11} & 0 & 0 & 0 \\ \times_{21} & \times_{22} & 0 & 0 \\ \times_{31} & 0 & 0 & 0 \\ \times_{41} & \times_{42} & \times_{43} & 0 \\ \times_{51} & \times_{52} & \times_{53} & 0 \end{bmatrix}.$$

Cela veut dire que les troisième et cinquième lignes de M sont linéairement dépendantes. À partir de L notons que les coefficients de la combinaison linéaire qui génère la cinquième ligne peuvent être récupérés en résolvant, via une procédure de subsitution arrière [33], le système triangulaire suivant

$$\begin{bmatrix} k_1 & k_2 & k_4 \end{bmatrix} \begin{bmatrix} \times_{11} & 0 & 0 \\ \times_{21} & \times_{22} & 0 \\ \times_{41} & \times_{42} & \times_{43} \end{bmatrix} - \begin{bmatrix} \times_{51} & \times_{52} & \times_{53} \end{bmatrix}, \tag{2.15}$$

et donc

$$\begin{bmatrix} k_1 & k_2 & 0 & k_4 & 1 \end{bmatrix} M = 0.$$

La procédure de substitution est également numériquement stable [43], et donc la qualité de la solution de (2.15) est aussi bonne que celle de la matrice calculée $\hat{L}$.

Notons qu'avec une permutation de lignes adéquate, la matrice L peut être mise sous la forme classique échelon par colonnes (Column Echelon Form, CEF). Parfois donc on dit que L est la CEF de M. On a utilisé cette notation dans quelques travaux comme [124, 126] issus de cette thèse. Finalement notons que par dualité,

la factorisation LQ de M^T est aussi la factorisation QR de M avec une stratégie de pivot par colonnes. Alors, on peut aussi définir une *forme échelon R* pour calculer une base naturelle de l'espace nul à droite de M.

2.3.4 Méthode généralisée de Schur

Comme on le démontrera dans les chapitres suivants, un des avantages de l'approche Toeplitz est la possibilité d'exploiter la structure matricielle. Une façon consiste à utiliser les méthodes basées sur la théorie du déplacement de rang [51] comme la méthode généralisée de Schur (*Generalized Schur Method*, GSM) [52]. Ici on présente brièvement les résultats qu'on utilise dans cette thèse.

Dans le cas particulier d'une matrice symétrique $M \in \mathbb{R}^{n\times n}$ on peut définir le déplacement (*displacement*)

$$\nabla M = M - FMF^T = X\Sigma X^T \tag{2.16}$$

avec

$$\Sigma = \mathrm{diag}\{I_p, -I_q\} = \begin{bmatrix} I_p & 0 \\ 0 & -I_q \end{bmatrix}$$

où p et q sont des dimensions appropriées. La matrice F qui est strictement triangulaire inférieure est appelée l'opérateur de déplacement (*displacement operator*). On appelle la matrice X le générateur symétrique (*symmetric generator*) de M. Le rang du déplacement (*displacement rank*) est $r = p + q = \mathrm{rank}(\nabla M)$.

Le générateur X n'est pas unique, n'importe quelle matrice T telle que $T^T\Sigma T = T\Sigma T^T = \Sigma$ donne un autre générateur XT. En particulier, on peut chercher une matrice T telle que le nouveau générateur ait la forme suivante

$$XT = \bar{X} = \begin{bmatrix} x_{11} & 0 \\ X_{21} & X_{22} \end{bmatrix} \tag{2.17}$$

où x_{11} est un scalaire et la matrice X_{22} est de dimension $(n-1)\times(r-1)$. On dit que $\bar{X}$ est un générateur propre (*proper generator*). Dans la section 4.4.2 de [51], un algorithme est décrit pour rendre propre un générateur quelconque. Cet algorithme se base sur l'application de matrices de Householder et de transformations hyperboliques [33]. Les générateurs propres ont un rôle clef dans la méthode généralisée de Schur. Il existe plusieurs versions de cette méthode, dans la section 7.4 de [51] on peut trouver une version très générale. Ici on utilise le résultat suivant.

Supposons qu'on ait un générateur symétrique propre X sous la forme (2.17) qui vérifie l'équation de déplacement (2.16). Alors, la matrice

$$\bar{M} = M - \begin{bmatrix} x_{11} \\ X_{21} \end{bmatrix} [x_{11}^T \quad X_{21}^T] = \begin{bmatrix} 0 & \\ & \bar{M}_{22} \end{bmatrix}$$

a ses premières colonne et ligne égales à zéro. La matrice $\bar{M}_{22}$ est le complément de Schur de M. Un générateur pour $\bar{M}$ est donné par $\bar{X} = [FX_{*1} \quad X_{*2}]$ où F est le même opérateur de déplacement que dans (2.16), c'est-à-dire

$$\bar{M} - F\bar{M}F^T = \bar{X}\Sigma\bar{X}^T. \tag{2.18}$$

Maintenant, comme F est strictement triangulaire inférieure, c'est-à-dire que sa première ligne est nulle, à partir de (2.18) on peut vérifier qu'en éliminant les premières ligne et colonne de $\bar{M}$ et la première ligne de $\bar{X}$ on obtient l'équation de déplacement

$$\bar{M}_{22} - \bar{F}\bar{M}_{22}\bar{F}^T = \bar{X}_2\Sigma\bar{X}_2^T,$$

où $\bar{F}$ s'obtient en éliminant les premières ligne et colonne de F. Donc, si on obtient une représentation propre de $\bar{X}_2$ (à l'aide de l'algorithme dans la section 4.4.2 de [51]), alors on peut obtenir le complément de Schur de $\bar{M}_{22}$. Avec cette procédure itérative, à la fin on peut obtenir la factorisation de Cholesky $M = LL^T$.

L'importance de la méthode généralisée de Schur est liée à la possible réduction d'un ordre de magnitude de la complexité algorithmique quand on factorise des matrices structurées. Formellement, on dit qu'une matrice est structurée si le rang de son déplacement est petit. Par exemple, le rang du déplacement d'une matrice Toeplitz est 2. Supposons que $M \in \mathbb{R}^{100\times100}$ soit Toeplitz et symétrique. Si on veut obtenir sa factorisation de Cholesky à l'aide de la méthode généralisée de Schur décrite ci-dessus, alors le nombre de colonnes du générateur est seulement 2. Evidemment le nombre d'opérations exécutées sera plus petit que si l'on considère les 100 colonnes de M.

Si la mis en oeuvre de la méthode généralisée de Schur est faite en considérant les 4 perfectionnements proposés dans la section 2.5 de [52], alors sa stabilité numérique peut être garantie pour le cas des matrices Toeplitz symétriques et définies positives.

Chapitre 3

Zero structure of polynomial matrices

Our objective in this chapter is to develop a numerical algorithm to obtain the finite and infinite structures of polynomial matrices. The theory of polynomial matrices is extendedly treated in the literature, so here we present only a few results formulated in a form and with a notation convenient for the sequel. Interested readers can consult mathematical books on polynomial matrices like [31]. In [12, 50, 109] some results on polynomial matrices related with systems theory are well explained.

Let $A(s) \in \mathbb{R}^{m\times n}[s]$ be a polynomial matrix with m rows and n columns, degree d and rank $r \leq \min(m, n)$. We write $A(s)$ as the matrix polynomial

$$A(s) = A_0 + A_1 s + \cdots + A_d s^d, \tag{3.1}$$

with coefficients $A_j \in \mathbb{R}^{m\times n}$ for $j = 1{:}d$ and s a complex indeterminate [1].

The structural information of polynomial matrices has important applications in control and systems theory. If a linear system is represented with polynomial matrices in the form (2.3), then the locations of its zeros and poles, which determine the dynamics of the system, are contained in the zero structure [2] of polynomial matrices $N(s)$ and $D(s)$ [50]. In the problem of decoupling of linear systems for example, infinite structural indices of a suitable polynomial matrix are needed to determine if the system is decouplable [21], and also to determine the structure that could have the decoupled closed loop system [122]. Model matching [67] and disturbance rejection [7] are other examples of control problems where the zero structure of polynomial matrices plays a fundamental role. The zero structure of a polynomial matrix is also instrumental to its spectral factorization which has applications in several optimal and robust control problems [34, 60], cf. Chapter 5 in this thesis.

1. All the results presented in this chapter are straightforwardly extended to complex-valued matrix coefficients.

2. The zero structure is also sometimes called the eigenstructure.

The classical tool to analyse the structure of a polynomial matrix is the canonical Smith form [50, 109].

Definition 3.1
Let $A(s)$ be as in (3.1). Its canonical *Smith form* is given by

$$S^A(s) = U(s)A(s)V(s) = \left[\begin{array}{ccc|c} f_1(s) & & & \\ & \ddots & & 0 \\ & & f_r(s) & \\ \hline & 0 & & 0 \end{array}\right] \tag{3.2}$$

where matrices $U(s)$ and $V(s)$ are *unimodular*[3]. Monic polynomials $f_i(s)$, $i = 1{:}r$, called the *invariant polynomials* of $A(s)$, are such that $f_i(s)$ divides $f_{i+1}(s)$. The roots of the invariant polynomials are called the *finite zeros* of $A(s)$. The number m_A of times that a given zero $s = z$ appears as a root in the invariant polynomials of $A(s)$ is its *algebraic multiplicity*. The number m_G of distinct invariant polynomials for which z is a root is its *geometric multiplicity*. ∎

From the above definition and for a given zero z, the Smith form (3.2) could be written as follows:

$$S^A(s) = S_z^A(s)\bar{S}_z(s) = \left[\begin{array}{ccccccc|c} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & 0 \\ & & & (s-z)^{k_1} & & & & \\ & & & & \ddots & & & \\ & & & & & & (s-z)^{k_{m_G}} & \\ \hline & & & 0 & & & & 0 \end{array}\right]$$
$$\left[\begin{array}{ccccccc|c} \bar{f}_1(s) & & & & & & & \\ & \ddots & & & & & & \\ & & \bar{f}_{r-m_G}(s) & & & & & 0 \\ & & & \bar{f}_{r-m_G+1}(s) & & & & \\ & & & & \ddots & & & \\ & & & & & & \bar{f}_r(s) & \\ \hline & & & 0 & & & & 0 \end{array}\right] \tag{3.3}$$

where $\bar{f}_i(s)$, $i = 1{:}r$ have no roots at $s = z$. Notice that integers k_i, $i = 1{:}m_G$, called here the *structural indices* associated to z are such that $k_i \leq k_{i+1}$. We say that $A(s)$ has a zero at z of order k_i. Matrix $S_z^A(s)$ will be called in this thesis the Smith form at z of $A(s)$.

Now let $\bar{U}(s) = U^{-1}(s)$ be the inverse of $U(s)$. Matrix $\bar{U}(s)$ is also polynomial because $U(s)$ is unimodular. In the following expressions, let $X_i(s)$ denote the ith

3. A polynomial matrix is unimodular if its determinant is a non-zero constant. Matrices $U(s)$ and $V(s)$ here gather the elementary row and column polynomial operations, cf. Example 2.5.

column of a matrix $X(s)$. From (3.2) and (3.3) we can check that

$$\begin{array}{rcl} A(s)V_1(s) & = & \bar{U}_1(s)\bar{f}_1(s) \\ & \vdots & \\ A(s)V_{r-m_G}(s) & = & \bar{U}_{r-m_G}(s)\bar{f}_{r-m_G}(s) \\ A(s)V_{r-m_G+1}(s) & = & \bar{U}_{r-m_G+1}(s)(s-z)^{k_1}\bar{f}_{r-m_G+1}(s) \\ & \vdots & \\ A(s)V_r(s) & = & \bar{U}_r(s)(s-z)^{k_{m_G}}\bar{f}_r(s) \\ A(s)V_{r+1}(s) & = & 0 \\ & \vdots & \\ A(s)V_n(s) & = & 0. \end{array} \tag{3.4}$$

From (3.4) notice that a finite zero z can also be defined as a value of s such that rank $A(z) < r$. The geometric multiplicity of z is $m_G = r - \text{rank } A(z)$, and the algebraic multiplicity is $m_A = k_1 + k_2 + \cdots + k_{m_G}$. Structural indices k_i, $i = 1{:}m_G$ are interpreted as the number of times that $A(s)V(s)$ has to be differentiated (w.r.t. s) in order that column $r - m_G + i$ becomes non-zero when evaluated at $s = z$. We summarize these results in the following Lemma. For more details see [4, 106] and the proof of Proposition 3.1 in this Chapter. Notice the analogy with the constant case of Jordan canonical forms [27], see also [31].

Lemma 3.1
Let $A(s)$ be as in (3.1). For a given finite zero z with geometric multiplicity $m_G = r - \text{rank}\, A(z)$, there exists structural indices $k_i > 0$, $i = 1{:}m_G$ such that the algebraic multiplicity of z is $m_A = k_1 + k_2 + \cdots + k_{m_G}$, and a series of vectors $v_{i1}, v_{i2}, \ldots, v_{ik_i}$, $i = 1{:}m_G$, called *eigenvectors* associated to z such that

$$\begin{bmatrix} \bar{A}_0 & \bar{A}_1 & \cdots & \bar{A}_{k_i-1} \\ & \bar{A}_0 & & \bar{A}_{k_i-2} \\ & & \ddots & \vdots \\ & & & \bar{A}_0 \end{bmatrix} \begin{bmatrix} v_{ik_i} \\ \vdots \\ v_{i2} \\ v_{i1} \end{bmatrix} = T_{k_i}\mathcal{V} = 0 \tag{3.5}$$

with $v_{11}, v_{21}, \ldots, v_{m_G 1}$ linearly independent and where

$$\bar{A}_j = \frac{1}{j!}\left[\frac{\mathrm{d}^j A(s)}{\mathrm{d}s^j}\right]_{s=z}. \tag{3.6}$$

Integer k_i is the length of the ith chain of eigenvectors associated to z, m_G is the number of chains of eigenvectors associated to z. Matrix T_{k_i} in (3.5) is in a block Toeplitz form. ∎

Definition 3.2
We define the *finite structure* of $A(s)$ as the set of finite zeros, their respective multiplicities and the chains of associated eigenvectors. ∎

Example 3.1
The polynomial matrix

$$A(s) = \begin{bmatrix} s^2 & 4 \\ 2-3s & s-6 \end{bmatrix}$$

has a Smith form $S^A(s) = \text{diag}\{1, s^3 - 6s^2 + 12s - 8\}$, so a finite zero at $s = 2$ with algebraic multiplicity 3 and geometric multiplicity 1. Now, from Lemma 3.1 and equation (3.5) it holds

$$\begin{bmatrix} 4 & 4 & 4 & 0 & 1 & 0 \\ -4 & -4 & -3 & 1 & 0 & 0 \\ 0 & 0 & 4 & 4 & 4 & 0 \\ 0 & 0 & -4 & -4 & -3 & 1 \\ 0 & 0 & 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 0 & -4- & 4 \end{bmatrix} \begin{bmatrix} -0.75 \\ 0 \\ 1 \\ 0 \\ -1 \\ 1 \end{bmatrix} = 0.$$

Hence, the 3 eigenvectors associated to $s = 2$ are

$$v_1 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \quad v_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad v_3 = \begin{bmatrix} -0.75 \\ 0 \end{bmatrix}.$$

■

Theoretically speaking, $s = \infty$ can also be an eigenvalue of $A(s)$. Traditionally the structure at infinity of $A(s)$ is analyzed via its canonical Smith MacMillan form at infinity.

Definition 3.3
Let $A(s)$ be as in (3.1). Its canonical *Smith MacMillan form at infinity* is given by

$$S^A_\infty(s) = U(s)A(s)V(s) = \left[\begin{array}{ccc|c} s^{-\gamma_1} & & & \\ & \ddots & & 0 \\ & & s^{-\gamma_r} & \\ \hline & 0 & & 0 \end{array}\right], \tag{3.7}$$

with possible negative integers $\gamma_i \leq \gamma_{i+1}$, $i = 1{:}r$ called structural indices at infinity and where $U(s)$ and $V(s)$ are biproper[4] rational matrices. If γ_i is negative, we say that $A(s)$ has a *pole at infinity* of order γ_i, if it is positive then we have a *MacMillan zero at infinity* of order γ_i. The sum denoted by

$$z_\infty = \sum_{i=t}^{r} \gamma_i, \tag{3.8}$$

with index t such that $\gamma_i > 0$ for $i \geq t$ is called the *MacMillan degree at infinity* (or number of MacMillan zeros at infinity) of $A(s)$. ■

4. A rational matrix is biproper if it is non-singular when $s \to \infty$. Matrices $U(s)$ and $V(s)$ here group the elementary row and column operations over the ring of rational functions.

In practice, in order to analyze the behavior at infinity, we use the change of variable $\bar{s} = s^{-1}$. Notice that $\bar{s} \to 0$ when $s \to \infty$, so the structure at infinity of $A(s)$ is related to the structure at zero of the dual polynomial matrix

$$A(\bar{s}) = A_0 + A_1\frac{1}{s} + \cdots + A_d\frac{1}{s^d} = \frac{1}{s^d}(A_0 s^d + A_1 s^{d-1} + \cdots + A_d) = \frac{1}{s^d}A_{\text{dual}}(s).$$

In other words,

$$S^A_\infty(s) = S^A_0(\bar{s}) = \frac{1}{s^d}S_0^{A_{\text{dual}}}(s)|_{s=s^{-1}}. \tag{3.9}$$

Notice that $A(\bar{s})$ is a rational matrix in the indeterminate s. The extension of the Smith form to rational matrices yields the Smith Macmillan form which allows to consider not only zeros but also poles[5] of the analyzed matrix [50, 109]. As a consequence, a polynomial matrix can be considered as a rational matrix whose poles are all located at infinity.

From (3.3), (3.7) and (3.9), we can check that the structure at infinity depends on the number and lengths of the chains of associated eigenvectors at $s = 0$ of the dual matrix $A_{\text{dual}}(s)$. Let $m_{G_\infty} = r - \operatorname{rank} A_d$ be the geometric multiplicity of $s = 0$ in $A_{\text{dual}}(s)$, and l_i, $i = 1{:}m_{G_\infty}$ be the structural indices at $s = 0$ in $A_{\text{dual}}(s)$. The structural indices at infinity γ_i, $i = 1{:}r$ are thus given by $\gamma_i = -d$ for $i = 1{:}r - m_{G_\infty}$, and $\gamma_{i+r-m_{G_\infty}} = l_i - d$ for $i = 1{:}m_{G_\infty}$. The algebraic multiplicity of $s = 0$ in $A_{\text{dual}}(s)$ is $m_\infty = l_1 + \cdots + l_{m_{G_\infty}}$. In the remainder of this thesis we simply consider that $A(s)$ has m_∞ *zeros at infinity* and that m_{G_∞} is the geometric multiplicity at infinity. This notation will be very useful in Chapter 5 for example. In the present Chapter, this notation allows us to define the structure at infinity by extending the results of Lemma 3.1.

Lemma 3.2
Let $A(s)$ be as in (3.1). Let $m_{G_\infty} = r - \operatorname{rank} A_d$ be the geometric multiplicity at infinity. Then there exists a series of integers $l_i > 0$ for $i = 1{:}m_{G_\infty}$ such that the algebraic multiplicity at infinity is $m_\infty = l_1 + l_2 + \cdots + l_{m_{G_\infty}}$. There also exists a series of eigenvectors at infinity $v_{i1}, v_{i2}, \ldots, v_{il_i}$ for $i = 1{:}m_{G_\infty}$ such that

$$\begin{bmatrix} A_d & A_{d-1} & \cdots & A_{d-l_i+1} \\ & A_d & & A_{d-l_i+2} \\ & & \ddots & \vdots \\ & & & A_d \end{bmatrix} \begin{bmatrix} v_{il_i} \\ \vdots \\ v_{i2} \\ v_{i1} \end{bmatrix} = T_{l_i}\mathcal{V} = 0 \tag{3.10}$$

with $v_{11}, v_{21}, \ldots, v_{m_{G_\infty}1}$ linearly independent. Integer l_i is the length of the ith chain of eigenvectors at infinity, m_{G_∞} is the number of chains of eigenvectors at infinity. Notice that block Toeplitz matrix T_{l_i} in (3.10) is obtained from (3.5) and (3.6) with $A_{\text{dual}}(s)$ and $z = 0$. ∎

Definition 3.4
We define the *infinite structure* of $A(s)$ as the multiplicities and the chains of associated eigenvectors at $s = 0$ of the dual matrix $A_{\text{dual}}(s) = A_d + A_{d-1}s + \cdots + A_0 s^d$.

∎

5. A pole is a value of the indeterminate for which a rational function tends to infinity.

Remark 3.1
From the m_∞ zeros at infinity of $A(s)$, only those corresponding to a chain of eigenvectors at infinity of length greater than d are also MacMillan zeros at infinity as in Definition 3.3. So, equation (3.8) giving the MacMillan degree at infinity of $A(s)$ can also be written as

$$z_\infty = \sum_{i=q}^{m_{G_\infty}} (l_i - d),$$

with index $q = t - r + m_{G_\infty}$ such that $l_i > d$ for $i \geq q$. ■

Finally, from Section 3.6 in [109], we extract the following result involving the structure of polynomial matrices. It will be useful in the remainder of this thesis.

Lemma 3.3
Let $A(s)$ be as in (3.1). Let p_∞ be the number of poles of $A(s)$ (it has only poles at infinity), m_f the number of finite zeros (including multiplicities) and z_∞ the number of MacMillan zeros at infinity (MacMillan degree at infinity), then

$$p_\infty = z_\infty + m_f + n_r + n_l$$

where n_r and n_l are, respectively, the sums of the degrees of the vectors in a minimal polynomial basis of the right and left null-spaces of $A(s)$ [6]. ■

Considering our definition of zeros at infinity as the zeros at $s = 0$ of the dual matrix $A_{\text{dual}}(s)$, we can easily reformulate the last Lemma as follows:

Corollary 3.1
Under the assumptions of Lemma 3.3, it holds

$$rd = m_\infty + m_f + n_r + n_l \tag{3.11}$$

where m_∞ is the number of zeros at infinity. ■

Example 3.2
The dual matrix of $A(s)$ in Example 3.1 is given by

$$A_{\text{dual}}(s) = \begin{bmatrix} 1 & 4s^2 \\ -3s + 2s^2 & s - 6s^2 \end{bmatrix}.$$

Its Smith form at $s = 0$ is $S_0^{A_{\text{dual}}}(s) = \text{diag}\{1, s\}$, so matrix $A(s)$ has one zero at infinity and, from (3.10), one eigenvector $w_1 = [0\ 1]^T$ at infinity such that $A_2 w_1 = 0$. Notice that equation (3.11) is verified with $m_\infty = 1$, $m_f = 3$ and $n_r = n_l = 0$. ■

6. For more details about minimal polynomial null-space bases, see Chapter 4.

Example 3.3
Here is an example of a control application of the concepts introduced in this section. Consider a linear system represented by the transfer matrix

$$\begin{bmatrix} s-3 & \alpha s^2+s+1 \\ 0 & s^2-1 \end{bmatrix} \begin{bmatrix} s^3 & 0 \\ 0 & s^3-1 \end{bmatrix}^{-1} = N(s)D^{-1}(s)$$

parametrized by a real scalar α. The above polynomial matrices are right coprime, so the finite poles and zeros of the system are, respectively, the finite zeros of $D(s)$ and $N(s)$ [50]. The system has therefore poles at $s = 0$ (triple), $s = 1$ and $s = -0.5 \pm 0.866i$. The Smith form of $N(s)$ is $S^N(s) = \text{diag}\{1, s^3 - 3s^2 - 2s + 6\}$, so the system has zeros at $s = 3$, $s = \pm 1.4142$. Now, suppose we want to determine the values of parameter α for which the system is decouplable by static state feedback. This is the case if and only if the sum of zeros at infinity is equal to the sum of zeros at infinity by rows [21, 118, 122]. The Smith form at 0 of $N_{\text{dual}}(s)$ is $S_0^{N_{\text{dual}}}(s) = \text{diag}\{1, s\}$. So, from (3.9), the Smith MacMillan form at infinity of $N(s)$ is $S_\infty^N(s) = \text{diag}\{s^2, s\}$. Finally, the Smith MacMillan form at infinity of $H(s)$ is given by $S_\infty^H(s) = s^{-3} S_\infty^N(s) = \text{diag}\{s^{-1}, s^{-2}\}$ which means that the system has zeros at infinity of orders 1 and 2 (see [50] for more details on the Smith MacMillan form of rational matrices). Following the same procedure for each row of $H(s)$, it is easy to see that the degrees of the rows of $N(s)$ have to be 1 and 2 for the system to be decouplable, i.e. $H(s)$ is decouplable by static state feedback if and only if $\alpha = 0$. ■

3.1 Computing the finite zeros

In the remainder of this Chapter we will consider that the locations of the finite zeros of the analyzed polynomial matrix are given. Note however that the problem of computing these zeros, referred to as the polynomial eigenvalue problem, is a very challenging problem in numerical computations. So we dedicate this short Section to review the classical algorithms as well as the mathematical tools for their analysis.

The standard method to compute the zeros of a polynomial matrix $A(s)$ is the application of the QZ algorithm over a linearization of $A(s)$ as a companion pencil $F(s) = F_1 s + F_0$, with F_1 the identity for monic polynomials. The QZ algorithm is a backward stable method which ensures that we compute the exact generalized eigenvalues of a slightly perturbed pencil $(F_1 + \Delta_1)s + (F_0 + \Delta_0)$ [33, 72]. In [101] the authors show that sometimes small perturbations Δ_1 and Δ_0 correspond to a large polynomial matrix perturbation $\Delta(s)$ in $A(s)$. In [23] it is proved, via a geometrical approach, how perturbations Δ_1 and Δ_0 yield first order perturbations in the coefficients of $A(s)$. Then exact expressions for the backward error $\Delta(s)$ are established. Based on these results, some scaling strategies on pencil $F(s)$ are developed in order to reduce the magnitude of $\Delta(s)$, see for example [63, 64]. It seems that the scaling over the pencil $F(s)$ corresponds to a change of the polynomial basis used to represent polynomial matrix $A(s)$. This scaling technique over the coefficients of $A(s)$ is illustrated for the quadratic eigenvalue problem in [24] for example. In [121] it is proved that a change of basis can improve the conditioning of a scalar polynomial.

The perturbation theory of the polynomial eigenvalue problem is studied in [19, 100]. Different expressions for the conditioning of a polynomial matrix are presented. The relation between these condition numbers and the conditioning of the associated pencil $F(s)$ have been recently studied in [46] and [66]. There it is shown that some linearizations are not convenient because the resulting pencil $F(s)$ is worse conditioned than the original matrix $A(s)$. Techniques to construct optimal linearizations, i.e. those that have condition numbers close to the condition number of $A(s)$, are also presented. This theory is limited to well-posed problems and as pointed out in [19], matrix polynomials with multiple roots can be infinitely ill-conditioned, or ill-posed. When multiple roots are considered, the analysis of the problem directly over the polynomials (numerical polynomial algebra [98]) allows to obtain satisfactory results. In [119] for example, a reliable numerical algorithm to compute multiple roots of scalar polynomials is presented.

Many other references, where the polynomial eigenvalue problem is treated from different points of view, can be cited. The extension of all these results to other problems with polynomial matrices is a line of promising research that has to be considered in the future in our opinion. In particular, we are interested in the extension of scaling techniques and change of basis as in [24, 121] for an arbitrary polynomial matrix $A(s)$. It seems to us that the application of the geometrical analysis in [119] to matrix polynomials has also critical importance. In the remainder of this thesis we propose some ideas of possible applications of these results to the problem of computing the zero structure or the null-space structure of polynomial matrices.

3.2 Algorithms

Following our notation of Section 2.3 and Algorithm 2.1, we distinguish our original polynomial problem (OPP) as the problem of finding the finite and infinite structure of a polynomial matrix $A(s)$ as in (3.1). As soon as the finite zeros are given, from Lemma 3.1 and 3.2 we recover the corresponding constant equivalent problems (CEP). If we assume that the lengths k_i, $i = 1{:}m_G$ and l_i, $i = 1{:}m_{G_\infty}$ are given, then solving block Toeplitz equations (3.5) and (3.10) yields the required associated eigenvectors. Now we analyze how to obtain these lengths. By duality (see equation (3.9)), notice that the analysis for a particular value of s can be extended to any value including infinity. So, in order to simplify the presentation, we develop our algorithm only for the infinite structure. Let matrix T_i be a block Toeplitz matrix with i block columns in the form

$$T_i = \begin{bmatrix} A_d & A_{d-1} & A_{d-2} & \cdots & A_{d-i+1} \\ & A_d & A_{d-1} & \cdots & A_{d-i+2} \\ & & A_d & & A_{d-i+3} \\ & & & \ddots & \vdots \\ & & & & A_d \end{bmatrix} \tag{3.12}$$

associated to polynomial matrix $A(s)$ with coefficients A_i as in (3.1).

Proposition 3.1
The following structural information of $A(s)$ is obtained by analyzing iteratively block Toeplitz matrices T_i of increasing dimensions. Let $r_0 = 0$, $r_1 = \operatorname{rank} T_1 = \operatorname{rank} A_d$ and $r_i = \operatorname{rank} T_i - \operatorname{rank} T_{i-1}$ for $i > 1$. Then

(a) Indices r_i satisfy $r_i \leq r_{i+1}$.
(b) Integer $x_i = r_{i+1} - r_i$ is the number of chains containing i eigenvectors at infinity.
(c) When $r_i = r$ for some $i = w$, then we have determined the lengths of all the chains of eigenvectors at infinity.
(d) If $r_w = r$ then $r_i = r$ for $i > w$.
(e) Index w is finite.

■

PROOF: First notice that matrix T_2 has the form

$$T_2 = \begin{bmatrix} T_1 & A_{d-1} \\ 0 & T_1 \end{bmatrix},$$

so, $r_2 = \operatorname{rank} T_2 - \operatorname{rank} T_1$ is equal to the number of linearly independent columns added in the second block column of T_2, that we call T_2^*. Suppose that $\operatorname{rank} T_1 = r_1$ and that the column k of T_1 is linearly independent. Then, because of the Toeplitz structure, the columns k and $k+n$ in T_2 are also linearly independent. On the other hand, if column j in T_1 is linearly dependent, it is possible that column $j+n$ in T_2, namely column j in T_2^*, is not. In other words $\operatorname{rank} T_2 \geq \operatorname{rank} T_1 + r_1$, i.e. $r_2 \geq r_1$. In general, matrix T_i can be written as

$$T_i = \begin{bmatrix} \boxed{T_{i-1}} & A_{d-i+1} \\ 0 & \boxed{T_{i-1}^*} \end{bmatrix}.$$

So, similarly, $r_i = \operatorname{rank} T_i - \operatorname{rank} T_{i-1}$ is equal to the number of linearly independent columns added in the rightmost block column of T_i that we call T_i^*. Using the same reasoning as above, it follows that $r_i \geq r_{i-1}$ which proves (a).

We know that $r \geq \operatorname{rank} A_i$, $i = 1{:}d$ (for more details see Chapter 4), so, $r_1 \leq r$. Now, from (3.4) one can see that the number n_0 of columns that become zero when evaluating at $s = 0$ is equal to $n_0 = n - r + m_{G_\infty}$. Comparing with (3.5), one can see that n_0 is also equal to the nullity of T_1, i.e. $n - r + m_{G_\infty} = n - r_1$ which confirms that $m_{G_\infty} = r - r_1$. Suppose that x_1 is the number of indices l_i, $i = 1{:}m_{G_\infty}$ equal to 1, namely the number of chains with only 1 eigenvector at infinity. If we take the first derivative in (3.4), x_1 columns can become non zero when evaluating at $s = 0$, so the number of columns that become zero is reduced to $n_1 = n - r + m_{G_\infty} - x_1$. Comparing with (3.5), one can see that n_1 is equal to the nullity of T_2 minus the

nullity of T_1, i.e. $n - r + m_{G_\infty} - x_1 = 2n - \operatorname{rank} T_2 - n + r_1 = n - r_2$ which confirms that $x_1 = r_2 - r_1$. Suppose that x_2 is the number of chains with only 2 eigenvectors at infinity. If we take the second derivative in (3.4), the number of columns that become zero when evaluating at $s = 0$ is only $n_2 = n - r + m_{G_\infty} - x_1 - x_2$. Comparing with (3.5), one can see that n_2 is equal to the nullity of T_3 minus the nullity of T_2, i.e. $n - r + m_{G_\infty} - x_1 - x_2 = 3n - \operatorname{rank} T_3 - 2n + \operatorname{rank} T_2 = n - r_3$ which confirms that $x_2 = r_3 - r_2$. In general, if x_i is the number of chains with i eigenvectors, from the ith derivative in (3.4) and comparing with (3.5), one can see that $x_i = r_{i+1} - r_i$ which proves (b).

Now suppose that the last chains have $w-1$ eigenvectors, so taking the $(w-1)$th derivative in (3.4), the number of columns that become zero when evaluating at $s = 0$ is only $n-r$, hence $x_1 + x_2 + \cdots + x_{w-1} = m_{G_\infty} = r - r_1$. Now using (b) to substitute x_i in the last equation, if follows that $r_w = r$ which proves (c). Point (d) is easily derived from (b) and (c): $x_i = 0$ for $i \geq w$, so $0 = r_{i+1} - r$.

Finally, from (3.11) we know that $m_\infty = l_1 + \cdots + l_{m_{G_\infty}}$ is finite, so no chain can have infinite length. In other words, $r_w = r$ for a finite index w which proves (e). Notice that $w \leq rd + 1$. □

From the last proposition and its proof, we can derive sufficient and necessary conditions for the absence of MacMillan zeros at infinity in a polynomial matrix.

Corollary 3.2
Let $A(s)$ be as in (3.1). Following the notations of Proposition 3.1, polynomial matrix $A(s)$ has no MacMillan zeros at infinity if and only if $r_{d+1} = r$. ■

PROOF: As pointed out in Remark 3.1, from the m_∞ zeros at $s = 0$ of the dual matrix $A_{\text{dual}}(s)$, only those corresponding to a chain of eigenvectors of length greater than d are also MacMillan zeros at infinity of $A(s)$. In other words from (b) of Proposition 3.1

$$z_\infty = r_{d+2} - r_{d+1} + 2(r_{d+3} - r_{d+2}) + \cdots + (w-1-d)(r - r_{w-1}) = \sum_{i=d+1}^{w-1} (r - r_i). \quad (3.13)$$

Now, from (d) and the last equation, notice that $r_{d+1} = r$ if and only if $z_\infty = 0$, which is the expected result, see also [40]. □

Now, based on Proposition 3.1, we sketch our general algorithm to compute the infinite structure of a polynomial matrix.

Algorithm 3.1 (`inf_gene`) *For a given polynomial matrix $A(s)$ as in* (3.1) *this algorithm computes the infinite structure, i.e. the multiplicities and the chains of eigenvectors associated to $s = 0$ in the dual matrix $A_{\text{dual}}(s)$, and the structural indices at infinity, i.e. the integers γ_i, $i = 1{:}r$ in* (3.7).

0. Let $\rho_0 = V = c = 0$, $r_0 = \operatorname{rank} T_1$, and $i = 1$.

i. With a rank revealing method (RRM) compute the rank ρ_i and a basis W of the null-space of T_i. Let $r_i = \rho_i - \rho_{i-1}$ and $x_{i-1} = r_i - r_{i-1}$. From V choose

x_{i-1} columns and arrange them in a matrix X. Now let $\mathcal{V}_{j+c} = X(:,j)$ and $l_{j+c} = i-1$ for $j = 1{:}x_{i-1}$ be the eigenvectors and the indices of the chains of length $i-1$.
If $r_i = r$, then go to step f1.
Finally let $c = c + x_{i-1}$, $V = W$ and $i = i+1$.

f1. *The chains of associated eigenvectors are $\mathcal{V}_i$, $i = 1{:}m_{G_\infty}$. The corresponding lengths are l_i, $i = 1{:}m_{G_\infty}$.*

f2. *The structural indices γ_i are as follows: $\gamma_i = -d$, $i = 1{:}r-m_{G_\infty}$ and $\gamma_{r-m_{G_\infty}+i} = l_i - d$, $i = 1{:}m_{G_\infty}$.*

Particular features of Algorithm 3.1 depend on the RRM used at each step. First we analyze the case where V is an orthogonal null-space basis obtained via the SVD of T_i for example. Notice that (3.10) is equivalent to

$$\begin{bmatrix} A_d & & & \\ A_{d-1} & A_d & & \\ \vdots & & \ddots & \\ A_{d-l_i+1} & A_{d-l_i+2} & \cdots & A_d \end{bmatrix} \begin{bmatrix} v_{i1} \\ v_{i2} \\ \vdots \\ v_{il_i} \end{bmatrix} = \bar{T}_{l_i} \bar{\mathcal{V}} = 0 \tag{3.14}$$

so, we can also use the LQ factorization of $\bar{T}_i$ to obtain an orthogonal basis $\bar{V}$ at each step[7]. For simplicity we consider that both the SVD and the LQ factorization can be applied to matrices $\bar{T}_i$. The algorithm reads as follows:

Algorithm 3.2 (`inf_orth`) *For a given polynomial matrix $A(s)$ as in* (3.1) *this algorithm computes the infinite structure and the structural indices at infinity.*

0. *Let $\rho_0 = V = c = 0$, $r_0 = \operatorname{rank} \bar{T}_1$, and $i = 1$.*

i. *With the SVD ($P^T \bar{T}_i Q = \Sigma$) or the LQ factorization ($\bar{T}_i Q = L$) compute the rank ρ_i and a basis $W = Q(:, \rho_i + 1 : in)$ of the null-space of $\bar{T}_i$. Let $r_i = \rho_i - \rho_{i-1}$ and $x_{i-1} = r_i - r_{i-1}$. From V choose x_{i-1} columns and arrange them in a matrix X. Now let $\bar{\mathcal{V}}_{j+c} = X(:,j)$ and $l_{j+c} = i-1$ for $j = 1{:}x_{i-1}$ be the eigenvectors and the indices of the chains of length $i-1$.*
If $r_i = r$, then go to step s1.
Finally let $c = c + x_{i-1}$, $V = W$ and $i = i+1$.

s1. *See steps f1 and f2 of* Algorithm 3.1.

Now, notice that along the lines of the proof of point (a) in Proposition 3.1, r_i can be obtained directly from the number of independent rows added in the last block row of $\bar{T}_i$. In other words, we do not have to factorize the whole matrix $\bar{T}_i$ but only its last block row. The general idea of this *blocked formulation* is illustrated as follows considering a matrix $A(s) \in \mathbb{R}^{3\times 3}[s]$. Let

$$\bar{T}_1 = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}$$

7. Notice that matrix $\bar{T}_i$ is the block transpose of matrix T_i in (3.12).

and suppose that we obtain an orthogonal matrix Q_1 such that

$$\bar{T}_1 Q_1 = \begin{bmatrix} \# & 0 & 0 \\ \# & \# & 0 \\ \# & 0 & 0 \end{bmatrix}.$$

Column $Q_1(:,3)$ is a basis of the null-space of $\bar{T}_1$. Now, notice that

$$\bar{T}_2 \begin{bmatrix} Q_1 & 0 \\ 0 & I_n \end{bmatrix} = \begin{bmatrix} \# & 0 & 0 & & & \\ \# & \# & 0 & & 0 & \\ \# & 0 & 0 & & & \\ \# & \# & \# & \times & \times & \times \\ \# & \# & \# & \times & \times & \times \\ \# & \# & \# & \times & \times & \times \end{bmatrix} = \mathcal{T}_2,$$

so, at step 2 we can process only the sub-matrix $\mathcal{T}_2(4:6,3:6)$. Suppose we obtain an orthogonal matrix $\bar{Q}_2$ such that

$$\mathcal{T}_2(4:6,3:6)\bar{Q}_2 = \begin{bmatrix} + & 0 & 0 & 0 \\ + & + & 0 & 0 \\ + & + & 0 & 0 \end{bmatrix}$$

then it follows that

$$\bar{T}_2 \begin{bmatrix} Q_1 & 0 \\ 0 & I_n \end{bmatrix} \begin{bmatrix} I_2 & 0 \\ 0 & \bar{Q}_2 \end{bmatrix} = \bar{T}_2 Q_2 = \left[\begin{array}{cc|ccccc} \# & 0 & 0 & & & & \\ \# & \# & 0 & & 0 & & \\ \# & 0 & 0 & & & & \\ \hline \# & \# & + & 0 & 0 & 0 \\ \# & \# & + & + & 0 & 0 \\ \# & \# & + & + & 0 & 0 \end{array}\right].$$

The columns $Q_2(:,5:6)$ then generate a basis of the null-space of $\bar{T}_2$. Moreover, r_2 is simply the rank of $\mathcal{T}_2(4:6,3:6)$, here equal to 2. The blocked formulation allows to reduce the number of operations required at each step. In fact, the number of rows of the analyzed matrices is always equal to m, and the number of columns can be smaller than in at each step. Next we describe the algorithm formally.

Algorithm 3.3 (`inf_orthblk`) *For a given polynomial matrix $A(s)$ as in* (3.1)*, this algorithm computes the infinite structure and the structural indices at infinity*[8].

0. Let $\rho_0 = V = c = 0$, $r_0 = \operatorname{rank} \bar{T}_1$, and $i = 1$.

i. With the SVD ($P^T T Q = \Sigma$) or the LQ factorization ($TQ = L$) compute the rank r_i and a basis $W = Q(:, r_i + 1 : in - \rho_{i-1})$ of the null-space of

$$T = \begin{bmatrix} \bar{T}_i((i-1)m+1 : im, 1 : (i-1)n)V & A_d \end{bmatrix}.$$

Let $\rho_i = \rho_{i-1} + r_i$, and $x_{i-1} = r_i - r_{i-1}$. From V choose x_{i-1} columns and arrange them in a matrix X. Now let $\bar{\mathcal{V}}_{j+c} = X(:,j)$ and $l_{j+c} = i - 1$ for

8. A similar algorithm to compute the structure of $A(s)$ at any point $s = z$ was presented in [106].

$j = 1{:}x_{i-1}$ be the eigenvectors and the indices of the chains of length $i-1$.
If $r_i = r$, then go to step s1.
Finally let $c = c + x_{i-1}$,

$$V = \begin{bmatrix} V & 0 \\ 0 & I_n \end{bmatrix} W,$$

and $i = i + 1$.

s1. See steps f1 and f2 of Algorithm 3.1.

Notice that at step i, any column of V can be a chain of $i-1$ eigenvectors at infinity, so in general, we have to choose x_{i-1} columns such that the vectors $v_{11}, v_{21}, \ldots, v_{c+x_{i-1}1}$ are linearly independent.

Now we extend Algorithm 3.1 to the case when $\mathcal{V}$ is a natural (non-orthogonal) null-space basis of T_i obtained via the R echelon form. The resulting algorithm reads as follows.

Algorithm 3.4 (`inf_eche`) *For a given polynomial matrix $A(s)$ as in* (3.1)*, this algorithm computes the infinite structure and the structural indices at infinity*[9]*.*

0. Let $\rho_0 = V = c = 0$, $r_0 = \operatorname{rank} T_1$, and $i = 1$.
i. Compute the R echelon form of T_i. Count the number ρ_i of linearly independent columns and compute a basis W of the null-space of R as described in Section 2.3.3. Let $r_i = \rho_i - \rho_{i-1}$ and $x_{i-1} = r_i - r_{i-1}$. From V choose x_{i-1} columns and arrange them in a matrix X. Now let $\mathcal{V}_{j+c} = X(:, j)$ and $l_{j+c} = i-1$ for $j = 1{:}x_{i-1}$ be the eigenvectors and the indices of the chains of length $i-1$.
If $r_i = r$, then go to step s1.
Finally let $c = c + x_{i-1}$, $V = W$ and $i = i + 1$.
s1. See steps f1 and f2 of Algorithm 3.1.

With this algorithm the orthogonal matrix Q is not computed, so, a blocked formulation as in Algorithm 3.3 is not possible. On the other hand, the advantage of this algorithm is that, from the R echelon form we can know the position of the linearly dependent columns in each block column of T_i. Therefore, we do not have to analyze matrices T_i for $i = 1, 2, 3, \ldots$. In fact, we can obtain the infinite structure only from the R echelon form of T_{rd+1}. The general idea is illustrated as follows considering a matrix $A(s) \in \mathbb{R}^{3\times 3}[s]$. Suppose that the R echelon form of T_1 is

$$H_1 T_1 = \begin{bmatrix} \times & \times & \times \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

and that columns 1 and 3 are linearly dependent, i.e. we pick the pivot in column 2. So, the corresponding null-space V computed as described in Section 2.3.3 has the form

$$V = \begin{bmatrix} 1 & 0 \\ k_{21} & k_{22} \\ 0 & 1 \end{bmatrix}.$$

9. A similar algorithm to compute the structure of $A(s)$ at any point $s = z$ was presented in [39].

Now, at step 2, we choose first the second column of T_2 as linearly independent and then we analyze the columns in the second block column of T_2. Suppose we obtain the factor

$$H_3H_2H_1T_2 = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & 0 \\ & & & 0 & 0 & 0 \\ & 0 & & 0 & 0 & 0 \\ & & & 0 & 0 & 0 \end{bmatrix}$$

and that column 6 is linearly dependent. So, the corresponding null-space V has the form

$$V = \begin{bmatrix} 1 & 0 & 0 \\ k_{21} & k_{22} & k_{23} \\ 0 & 1 & 0 \\ 0 & 0 & k_{43} \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Notice that the null-space of T_1 is recovered in the first 2 columns of V. The third column of V corresponds to the chain with 2 or more eigenvectors at infinity. Finally, suppose that at step 3, after taking columns 2, 4 and 5 of T_3 as independent, we obtain the factor

$$H_6H_5\cdots H_1T_3 = \begin{bmatrix} \times & \times & \times & \times & \times & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & 0 & \times & \times & \times \\ & & & 0 & 0 & 0 & \times & \times & \times \\ & 0 & & 0 & 0 & 0 & 0 & \times & \times \\ & & & 0 & 0 & 0 & 0 & \times & 0 \\ & & & & & & 0 & 0 & 0 \\ & & & 0 & & & 0 & 0 & 0 \\ & & & & & & 0 & 0 & 0 \end{bmatrix}$$

which has the null-space basis

$$V = \begin{bmatrix} 1 & 0 & 0 \\ k_{21} & k_{22} & k_{23} \\ 0 & 1 & 0 \\ 0 & 0 & k_{43} \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ & & \\ & 0 & \\ & & \end{bmatrix}.$$

Now notice that no new columns are added to the null-space, so $r_3 = r$. Notice also that we can easily recover the chains of eigenvectors from this last null-space basis V. The first chain with only 1 eigenvector is given by $V(1:3,1)$. The second chain with

2 eigenvectors is given by $V(1:6,3)$. In conclusion, computing only the R echelon form of T_{rd+1} we can obtain all the structure at infinity of $A(s)$. If $A(s)$ has chains of eigenvectors of length near to rd, it is clear that computing only the R echelon form of T_{rd+1} will be more efficient than analyzing matrices T_i for $i = 1{:}rd$. On the other hand, if all the chains has only few eigenvectors and d is large, computing the R echelon form of T_{rd+1} could require an unnecessarily large computational effort. A compromise between both approaches must be found in order to perform the minimum number of steps. Here we propose to analyze at step i in Algorithm 3.4 the matrix T_j for $j = j + \kappa$ with $\kappa = \kappa + \gamma rd$ where γ is a fixed ratio. Next we describe the algorithm formally.

Algorithm 3.5 (`inf_echeblk`) *For a given polynomial matrix $A(s)$ as in* (3.1)*, this algorithm computes the infinite structure and the structural indices at infinity.*

0. *Let $j = 0$, $c = 0$ and $\kappa = 1$.*

loop. *Let $j = \mathrm{int}(j + \kappa)$ where $\mathrm{int}(x)$ is the nearest integer to x. Let $\kappa = \kappa + \gamma rd$. Compute the R echelon form of T_j, choosing the linearly independent columns in each block column from the left to the right. Count the number r_t $t = 1{:}j$ of linearly independent columns in each block column of T_j. If $r_t = r$, then go to step s1.*

s1. *For $i = 1{:}t-1$, let $x_i = r_{i+1} - r_i$ and compute the linear combination of the x_i columns which are dependent in the ith block column of R and independent in the $(i+1)$th block column of R and arrange them in X. Now let $\mathcal{V}_{j+c} = X(:,j)$ and $l_{j+c} = i$ for $j = 1{:}x_i$ be the eigenvectors and the indices of the chains of length i. Finally let $c = c + x_i$.*

s2. *See steps f1 and f2 of* Algorithm 3.1.

3.3 Algorithmic analysis

In this section we analyse the performance of the different algorithms presented above. The algorithms are analyzed in two aspects, computational complexity and numerical stability. Another objective of this section is to compare our algorithms with the standard algorithms in the literature to compute the zero structure of a polynomial matrix. These standard algorithms are based on the linearization of the polynomial matrix (the pencil approach, cf. Section 2.3 and [105, 107]). Next we use these results to reformulate a simple method to compute the infinite structural indices of polynomial matrices which we compare with our algorithms. For more details see Algorithm 3.6 in [105] and Algorithm 1 in [107].

Algorithm 3.6 (`inf_pencil`)

0. Given $A(s)$ as in (3.1), first build the linearization

$$P(s) = sE - L = s \begin{bmatrix} & -I_n & & \\ & & \ddots & \\ & & & -I_n \\ A_0 & A_1 & \cdots & A_{d-1} \end{bmatrix} - \begin{bmatrix} -I_n & & & \\ & \ddots & & \\ & & -I_n & \\ & & & -A_d \end{bmatrix} \tag{3.15}$$

Let $L_1 = L$, $E_1 = E$, $\rho_0 = nd$ and $i = 1$.

i. Compute the SVD ($P^T L_i Q = \Sigma$) and the rank ρ_i of L_i.
If $v_i = \rho_{i-1} - \rho_i$ is zero then take $w = i$ and go to End.
Make the column compressions $L_i Q = [L\ 0]$, and $E_i Q = [E\ N]$, where E and L have ρ_i columns. Compute the SVD ($P^T N Q = \Sigma$) of N, and make the row compressions

$$MP^T L = \begin{bmatrix} \times \\ L_{i+1} \end{bmatrix}, \qquad MP^T E = \begin{bmatrix} \times \\ E_{i+1} \end{bmatrix}$$

where L_{i+1} and E_{i+1} have ρ_i rows, and M is a suitable permutation matrix. Let $i = i + 1$.

End. For $i = 1{:}w-1$, $A(s)$ has $x_i = v_i - v_{i+1}$ chains of eigenvectors at infinity of length i.

3.3.1 Complexity

First we analyse the expressions of complexity, or number of flops, of the presented algorithms. Several parameters enter the final expressions. We illustrate this when analysing the complexity of Algorithm 3.3 and when comparing it with the pencil algorithm sketched above, see also [123]. For simplicity, we consider only the case of square full-rank $n \times n$ polynomial matrices. We suppose that the SVD of a matrix $M \in \mathbb{R}^{k \times l}$ requires $4k^2l + 8kl^2 + 9l^3$ operations, and that a multiplication $C = MN$ with $M \in \mathbb{R}^{k_A \times l}$ and $N \in \mathbb{R}^{l \times k_B}$ requires $(2l-1)k_A k_B$ operations [33].

Denote by $\rho = r_1 + r_2 + \cdots + r_{i-1}$ the rank of $\bar{T}_{i-1}$. So, at step i, Algorithm 3.3 performs one multiplication of dimension $n \times (i-1)n$ by $(i-1)n \times ((i-1)n - \rho)$ to compute T, i.e.

$$\text{flops}_{T_1} = -n[2ni - (2n+1)]\rho + n^2[2ni^2 - (4n+1)i + (2n+1)],$$

one SVD of dimension $n \times (in - \rho)$ to compute the rank and null-space of T, i.e.

$$\text{flops}_{T_2} =, -9\rho^3 + n[27i + 8]\rho^2 - n^2[27i^2 + 16i + 4]\rho + n^3[9i^3 + 8i^2 + 4i]$$

and finally one multiplication of dimensions $(i-1)n \times (in-\rho)$ by $(in-\rho) \times (in-\rho-r_i)$ to compute $\bar{V}$, i.e.

$$\begin{aligned} \text{flops}_{T_3} = {} & 2n[i-1]\rho^2 + n[-4ni^2 + (4n + 2r_i + 1)i - (2r_i + 1)]\rho + \\ & n[2n^2 i^3 - n(r_i + 2n + 1)i^2 + (2nr_i + r_i + n)i - r_i]. \end{aligned}$$

Then if $r_w = n$, the total number of flops performed by the algorithm is given by

$$\text{flops}_T = \sum_{i=1}^{w} \text{flops}_{T_1} + \sum_{i=1}^{w} \text{flops}_{T_2} + \sum_{i=1}^{w-1} \text{flops}_{T_3}. \tag{3.16}$$

Now let $\nu = v_1 + v_2 + \cdots + v_{i-1}$. So, at step i, the pencil algorithm 3.6 performs one SVD of dimensions $(nd-\nu) \times (nd-\nu)$ to compute the rank and the right null-space of L_k, i.e.

$$\text{flops}_{P_1} = -21\nu^3 + 63nd\nu^2 - 63n^2d^2\nu + 21n^3d^3$$

one multiplication of dimensions $(nd-\nu) \times (nd-\nu)$ by $(nd-\nu) \times (nd-\nu)$ to column compress E_k, i.e.

$$\text{flops}_{P_2} = -2\nu^3 + [6nd-1]\nu^2 - 2n[3nd^2-d]\nu + n^2[2nd^3-d^2],$$

another SVD of dimensions $(nd-\nu) \times v_i$ to compute the left null-space of N, i.e.

$$\text{flops}_{P_3} = 4v_i\nu^2 - 8v_i[nd+v_i]\nu + v_i[4n^2d^2 + 8nv_id + 9v_i^2],$$

and finally 2 multiplications of dimensions $(nd-\nu)\times(nd-\nu)$ by $(nd-\nu)\times(nd-\nu-v_k)$ to row compress E and L, i.e.

$$\text{flops}_{P_4} = -4\nu^3+2[6nd-2v_i-1]\nu^2-2[6n^2d^2-2n(2v_i+1)d+v_i]\nu+2n[2n^2d^3-n(2v_i+1)d^2+v_id],$$

Then if $v_w = 0$, the total number of flops performed by the pencil algorithm is given by

$$\text{flops}_P = \sum_{i=1}^{w} \text{flops}_{P_1} + \sum_{i=1}^{w-1} \text{flops}_{P_2} + \sum_{i=1}^{w-1} \text{flops}_{P_3} + \sum_{i=1}^{w-1} \text{flops}_{P_4}. \tag{3.17}$$

So notice from equations (3.16) and (3.17) that several quantities are involved, not only the dimension n of the matrix, but also its degree d and in general its structure, namely nullities v_k, indices r_i and number of steps w. In order to study these expressions, we consider first that the indices r_i and the number of steps are fixed and we analyze the dependence on n and d. Then, we consider the general case, namely, when the infinite structure of $A(s)$ is not fixed.

Dependence on degree d and dimension n for a given structure

It is easy to show that $v_i = n - r_i$, so if we consider indices r_i, $i = 1{:}w$ fixed, then nullities v_i are also fixed. In that case, from (3.17) notice that the algorithmic complexity of the pencil algorithm is $O(n^3d^3)$, see also [107]. On the other hand, from (3.16) we can see that dependence of degree d is eliminated, the complexity of Algorithm 3.3 is $O(n^3)$.

Example 3.4
Consider the polynomial matrix of degree d

$$A(s) = \begin{bmatrix} s^d & p_1(s) & p_2(s) \\ 0 & s^{d-a} & p_3(s) \\ 0 & 0 & s^{d-a-b} \end{bmatrix}, \tag{3.18}$$

d	inf_orthblk	inf_pencil
20	136683	37181275
40	136683	349798015
60	136683	1246161955
80	136683	3034721095

TAB. 3.1 – *Number of flops executed by the different algorithms when computing the infinite structure of matrix* (3.18) *for* $a = 5$, $b = 2$ *and different values of* d.

where a, b are given integers and $p_1(s)$, $p_2(s)$, $p_3(s)$ are given polynomials. If the degree of $p_3(s)$ is less than $d - a$, then we can see that the vector of indices r_i is given by

$$r = [\underbrace{1, 1, \ldots, 1}_{a}, \underbrace{2, 2, \ldots, 2}_{b}, 3],$$

so, the vector of the nullities v_i is

$$v = [\underbrace{2, 2, \ldots, 2}_{a}, \underbrace{1, 1, \ldots, 1}_{b}, 0].$$

Now suppose we fix the structure of $A(s)$ with $a = 5$ and $b = 2$ and we vary the degree d in order to show how it affects the number of performed flops. The results are presented in Table 3.1. We can see that the algorithmic complexity of the Toeplitz Algorithm 3.3 does not depend directly on the degree of the matrix. On the other hand, we can see the rapid growth in the number of operations executed by the pencil Algorithm 3.6 when degree d increases. ∎

Example 3.5
Now consider the polynomial matrix

$$A(s) = \begin{bmatrix} s^3 I_b & & & \\ & 1 & s^3 & 0 \\ & 0 & 1 & s \\ & 0 & 0 & 1 \end{bmatrix} \tag{3.19}$$

where b is a given integer. In this case we can increase b in order to increase the dimension n of $A(s)$, keeping the degree and the vectors r and v constants. The results are presented in Table 3.2. Now we see that algorithmic complexities of both algorithms depend on dimension n. ∎

From these results we can conclude that one advantage of the Toeplitz approach is the independence on the matrix degree in some specific cases. Nevertheless, notice that in general a change on the degree or the dimension of a polynomial matrix modifies its structure at infinity. Next we analyze this general case.

b	inf_orthblk	inf_pencil
17	2518093	39830120
37	14935813	362273240
57	45335133	1275721160
77	101780053	3088621880

TAB. 3.2 – *Number of flops executed by the different algorithms when computing the infinite structure of matrix* (3.19) *for different values of* b.

General case: dependence on structural indices

From equations (3.16) and (3.17) we see that large indices r_i, $i = 1{:}w$ are favourable for the Toeplitz algorithm because the growth of Toeplitz matrices at each step is smaller. At the same time, since $r_i = n - v_i$, this case is unfavourable for the pencil algorithm because the deflation at each step is also smaller. In contrast, if we have small indices r_i, the Toeplitz matrices analyzed at each step are larger, but the matrices analyzed by the pencil algorithm are smaller.

Now, considering the number of steps, it is clear that both algorithms are faster if w is small. Nevertheless, notice that w small is more favourable for the Toeplitz algorithm than for the pencil algorithm which starts always with the larger pencil $P(s)$ in (3.15). In other words, for the pencil algorithm to execute less operations than the Toeplitz algorithm, a sufficiently large number of steps is necessary. The number of steps depends on the structure of the analyzed matrix as described in Corollary 3.1. From (3.11) we can see that the maximum number of steps is $w_{\max} = nd + 1$, but this bound is attained only if the geometric multiplicity at infinity $v_1 = n - r_1$ is one, namely, if there is only one chain of eigenvectors at infinity. If v_1 is large, the maximum number of steps is reduced, in general, $w_{\max} = nd + 1 - (v_1 - 1)$. So, a large number of steps is also more favourable for the Toeplitz algorithm because a small v_1 implies large indices r_i.

As a general conclusion we say that the type of matrices for which the pencil Algorithm 3.6 is more efficient than the Toeplitz Algorithm 3.3 are matrices with a small degree d but with a structure that implies a number of steps w close to w_{max}, namely, matrices with few chains of eigenvectors but with large length. Nevertheless, the number of operations performed by both algorithms in these cases will be comparatively large. On the other hand, we can easily find a polynomial matrix very favourable for the Toeplitz algorithm and simultaneously very unfavourable for the pencil algorithm. We illustrate that in the following example.

Example 3.6
Consider the 40×40 polynomial matrix

$$A(s) = \begin{bmatrix} 1 & s^2 & & 0 \\ & \ddots & \ddots & \\ & & \ddots & s^2 \\ 0 & & & 1 \end{bmatrix}$$

which has one zero at infinity of order 80. Vectors v and r are

$$v = [\underbrace{1, 1, \ldots, 1}_{80}, 0], \quad r = [\underbrace{39, 39, \ldots, 39}_{80}, 40].$$

The number of operations executed by the algorithms are $\text{flops}_P = 324244800$ and $\text{flops}_T = 2052129600$. The ratio between this number of operations is

$$\frac{\text{flops}_T}{\text{flops}_P} = 6.33.$$

So, notice that although the pencil algorithm is more efficient, the number of operations that both algorithms execute is comparatively large. On the other hand, consider the matrix

$$A(s) = \begin{bmatrix} s^{40} & & \\ & s^{39} & \\ & & s^{39} \end{bmatrix}$$

which has 2 zeros at infinity of order 1. Vectors v and r are

$$v = [2, 0] \quad r = [1, 3].$$

The number of operations are $\text{flops}_P = 87946704$ and $\text{flops}_T = 2502$. The ratio between this number of operations is

$$\frac{\text{flops}_P}{\text{flops}_T} = 35150.56.$$

Now, notice the great difference between the performances of both algorithms. ■

The Toeplitz algorithm can be considered, therefore, as an alternative, if not a competitor, to the classical pencil algorithm. At least, as far as complexity is concerned, we have seen that the Toeplitz algorithm is more efficient in most of the cases. Also it is important to remark that, with the same computational effort, the Toeplitz algorithm returns not only the structural indices but also the associated eigenvectors, which is not the case for the pencil algorithm. On the other hand, the pencil algorithm returns some other information. First notice that when applying the pencil algorithm, we do not have to know the rank of $A(s)$, in fact this information is a sub-product of the algorithm. Second, for the case of singular matrices, after the pencil algorithm stops, it can be shown that not only the structural infinite indices are obtained but also the right null-space indices [105, 107], cf. Chapter 4 of this

thesis. Moreover, it can be shown that the unprocessed part of pencil (3.15) contains only the information on the finite structure and the left null-space of $A(s)$. In the next chapter, we show how the Toeplitz approach can also be extended to compute all the eigenstructure and the rank of polynomial matrices, see also [124].

Now, how does the Toeplitz algorithm perform with respect to the other algorithms presented in Section 3.2? We answer this question next. First notice that when using the SVD ($P^T T Q = \Sigma$) to compute the rank and the null-space of T at each step in Algorithm 3.3, half of the computational effort is wasted because orthogonal matrix P is never used. So, we can expect a better performance applying the more economical LQ factorization. Now, we compare Algorithm 3.3 using the LQ factorization with Algorithm 3.5. We have seen that a large number of steps, which requires large indices r_i, affects the performance of Algorithm 3.3 but at the same time, we have noted that large indices r_i imply that the final number of required flops is not very large. With Algorithm 3.5 we can reduce directly the number of steps even when the analyzed matrix has large chains of associated eigenvectors. This reduction on the number of steps depends on the choice of ratio γ. Unfortunately, we have not a method to determine the optimal value of γ and this could have a negative impact on the number of executed flops as we illustrate in the following example.

Example 3.7
Consider the polynomial matrix

$$A(s) = \begin{bmatrix} 1 & s^4 & s \\ 0 & 1 & s^{50} \\ 0 & 0 & 1 \end{bmatrix}$$

which has two chains of eigenvectors at infinity, one of length 46 and the other of length 104. So, Algorithm 3.3 stops after 105 steps. The execution time required by this algorithm is 4.81 sec[10]. On the other hand, the execution time required by Algorithm 3.5 depends on the number of steps and the dimension of the Toeplitz matrices analyzed at each step which is controlled by the ratio γ. In Table 3.3 we present the execution times in seconds for different values of γ. We also report the values that j, the number of block columns of the analyzed block Toeplitz matrix, takes at each step. Notice that for this example, $\gamma = 0.1$ is the worst option because the algorithm execute 5 steps including one with the largest possible Toeplitz matrix T_{151}. Values $\gamma = 0.3$ or 0.9 are better options because the number of steps is reduced but also the largest analyzed Toeplitz matrix has only 138 block columns. The optimal option here is $\gamma = 0.69$ which implies only 2 steps and a largest Toeplitz matrix with exactly 105 block columns which is the minimum necessary to find the vectors in the largest chain of eigenvectors. Obviously we cannot know the optimal value of γ from the simple observation of the dimension and the degree of $A(s)$. In conclusion, Algorithm 3.5 should be applied carefully when a small number of executed flops is required. ■

10. Using a SUN microsytems Ultra 5 with Matlab 7 and with the algorithm programmed from scratch, namely, without compiled built-in functions such as `qr`.

γ	time	j
0.1	6.99	1,17,48,94,151
0.3	4.99	1,47,138
0.5	6.07	1,77,151
0.69	**3.02**	**1,105**
0.9	4.8	1,137

TAB. 3.3 – *Execution times in seconds and number j of block columns in the Toeplitz matrices analyzed at each step by* Algorithm 3.5 *for different values of γ.*

3.3.2 Stability and accuracy

In this section we analyze the stability of Algorithms 3.2 and 3.3, but the analysis can be easily extended to Algorithms 3.4 and 3.5. We start analyzing the stability of the CEP, namely, the stability of the numerical methods used to obtain the rank and the different constant null-spaces of the Toeplitz matrices $\bar{T}_i$ at each step. In Sections 2.3.1 and 2.3.2 we have seen that the SVD and the LQ factorization are backward stable methods that compute, with a suitable accuracy, the rank and the null-space of a constant matrix. In that way, backward stability of the SVD or the LQ factorization in Algorithm 3.2 is guaranteed. On the other hand, showing the backward stability of the blocked SVD or LQ factorization used in Algorithm 3.3 is not difficult. In fact we only have to consider the blocked process described after Algorithm 3.2 as a sequence of some particular Householder transformations and then apply the proof of Theorem 18.3 in [43]. For simplicity we illustrate this on a 4×3 polynomial matrix $A(s)$ using the LQ factorization. The extension to the general case is direct.

Consider the computed LQ factorization $\hat{L}_1 = (\bar{T}_1 + \Delta)\hat{H}_1\hat{H}_2 = (\bar{T}_1 + \Delta)\hat{Q}_1$ with

$$\hat{L}_1 = \begin{bmatrix} \# & 0 & 0 \\ \# & 0 & 0 \\ \# & \# & 0 \\ \# & \# & 0 \end{bmatrix}, \quad \|\Delta\|_2 \leq \phi_1 \|\bar{T}_1\|_2.$$

We represent by $\hat{X}$ the computed matrix of X. Here ϕ_1 is a small constant depending on the dimensions of $\bar{T}_1$, the machine precision ε, and the number of applied Householder reflections. Now suppose that we want to obtain the LQ factorization of $\bar{T}_2$ using the blocked process of Algorithm 3.3. We start with

$$(\bar{T}_2 + \Delta) \begin{bmatrix} \hat{Q}_1 & 0 \\ 0 & I_n \end{bmatrix}.$$

Here the product of the second block column of $\bar{T}_2$ by the identity does not have to be computed and so, the second block column of Δ can be simply set to zero. Nevertheless, the dimension has changed, so we write $\|\Delta\|_2 \leq \bar{\phi}_1 \|\bar{T}_2\|_2$ where $\bar{\phi}_1$ depends on the dimension of $\bar{T}_2$, so it represents a more pessimistic upper bound

than ϕ_1. Then we apply a third and a fourth Householder reflection to obtain

$$\hat{L}_2 = (\bar{T}_2 + \Delta) \begin{bmatrix} \hat{Q}_1 & 0 \\ 0 & I_n \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \hat{H}_3 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \hat{H}_4 \end{bmatrix} = (\bar{T}_2 + \Delta)\hat{Q}_2,$$

and so we write $\|\Delta\|_2 \leq \phi_2 \|\bar{T}_2\|_2$. This bound for Δ is even more pessimistic. This can be expected because we have performed two more Householder reflections. We summarize these results in the following lemma.

Lemma 3.4
Consider that a polynomial matrix $A(s)$ has a chain of eigenvectors of length k. The application of the blocked LQ factorization (as in Algorithm 3.3) to solve system (3.14) is backward stable, namely

$$(\bar{T}_k + \Delta)\hat{\bar{\mathcal{V}}} = 0, \qquad \|\Delta\|_2 \leq \phi \|\bar{T}_k\|_2,$$

where $\hat{\bar{\mathcal{V}}}$ is the chain of computed eigenvectors and ϕ is a small constant depending on the dimensions of $\bar{T}_k$, the machine precision ε, and the number of applied Householder reflections. ∎

Now the question is how close is the computed solution $\hat{\bar{\mathcal{V}}}$ from the real solution? or even more importantly, are the obtained number of chains and the respective lengths correct? It is well known that the rank estimation is an ill-posed problem, so we can expect that our OPP is also ill-posed. In other words, a small perturbation in the coefficients of $\bar{T}_k$ (perturbation in the coefficients of $A(s)$) can yield an important change in the obtained structure. We can pass from 2 chains to 3 chains of eigenvectors for example. Some strategies to render the problem well-posed can be formulated along the lines in [119], cf. Section 3.1, see also [97, 98]. Given a first approximation of the structure at infinity of $A(s)$, we can think of some kind of iterative refinement over some manifold where this structure is invariant (the problem is well-posed). Other possible way to improve the conditioning (posedness) of the matrix can be a scaling along the lines in [24, 121]. The extension of all these results, however, is out of the scope of this thesis and could be a subject of future work.

While these results are available, at least we must make sure that the information obtained at each step by the algorithms is what we expect. With infinite precision, the rank of T_i computed with only one SVD or LQ factorization is equal to the rank computed using the blocked process of Algorithm 3.3. Nevertheless, with finite precision this could be no more true. In that case Algorithms 3.2 and 3.3 will give different results. We illustrate this with the following academic example.

Example 3.8
Consider the full-rank polynomial matrix

$$A(s) = \begin{bmatrix} 10^{-8}s & 10^{-8}s^2 & 1 \\ 20 & 10s & 0 \\ 0 & 1+20s & 10^8 \end{bmatrix}$$

Matrix $T_1 = A_2$ has exact rank 1. The rank computed via the SVD or the LQ factorization with Matlab is also 1 (a pivot is chosen in the first row), so $\hat{r}_1 = 1$. Now, looking at matrix T_2, we can see that the exact rank is 2. The rank obtained with Matlab is also 2, so $\hat{r}_2 = 1$. Notice, that the LQ factorization in this case picks pivots in rows 6 and 4. Row 1 is now dependent because the pivot was chosen in row 6 which has a larger norm. At step 3, while the exact rank of T_3 is 5, the rank computed by Matlab is only 4, so $\bar{r}_3 = 2$. Now the pivot in row 4 is also lost, but there is no other row to replace it. On the contrary, the exact value of r_3 is $\operatorname{rank} A(s) = 3$ which means that $A(s)$ has 2 chains of 2 eigenvectors at infinity. Finally, when computing the rank of T_4 we recover $\bar{r}_4 = 3$. This means that $A(s)$ has one computed chain of 2 eigenvectors and another one with 3 eigenvectors, namely 5 computed zeros at infinity, which is wrong. On the other hand, using the blocked process of Algorithm 3.3, at step 3 we analyze only the 3 bottom rows of $\mathcal{T}_3$. The pivots in rows 1 and 4 are kept and 3 more pivots (in linearly independent rows 7, 8 and 9) are found, so we recover $\bar{r}_3 = 3$ which is the correct value. Now suppose we balance the norms of A_i, $i = 0{:}2$ using the results in [24]. The balanced matrix is

$$\bar{A}(s) = \begin{bmatrix} 8.8 \times 10^{-10} s & 0.028 s^2 & 2.8 \times 10^{-9} \\ 5.6 \times 10^{-8} & 0.88 s & 0 \\ 0 & 2.8 \times 10^{-9} + 1.8 s & 0.028 \end{bmatrix}.$$

With this matrix, application of the SVD or the LQ factorization to matrix T_3 yields $\bar{r}_3 = 3$ which is the expected value. Moreover notice that rows chosen by the LQ factorization to select a pivot are (in order): 9, 6, 7, 8 and 4. The pivot in row 1 is replaced by row 6 which can improve numerics, but also the pivot in row 4 is kept which allows to recover the correct structure of $A(s)$. Unfortunately this scaling is not yet extended to matrices of higher degree, cf. Section 3.1. In conclusion, while no regularization techniques are available for the problem of computing the zero structure of an arbitrary polynomial matrix $A(s)$, we recommend the use of Algorithm 3.3. ∎

Now, let the lengths and the number of chains of eigenvectors be fixed. Accuracy of the computed eigenvectors is then ensured by the LQ factorization or the SVD. In fact, a vector $\hat{\bar{\mathcal{V}}}$ is chosen because $\bar{T}_i \hat{\bar{\mathcal{V}}}$ is suitably small, cf. Sections 2.3.1 and 2.3.2. Moreover, the forward error $\|\bar{\mathcal{V}} - \hat{\bar{\mathcal{V}}}\|$ has a magnitude of the order of the backward error Δ introduced by the factorization [42, 43, 99], and this backward error is small, cf. Lemma 3.4. However, in general backward error Δ has no block Toeplitz structure, in other words it is not possible to ensure that $\hat{\bar{\mathcal{V}}}$ is the exact chain of eigenvectors of a slightly perturbed matrix $A(s)$. The analysis has to go back to the coefficients of $A(s)$ as we noted in Section 2.3 of this thesis. We present this analysis in Section 4.2.2 for the problem of computing null-spaces of polynomial matrices, but these results are straightforwardly extended to the zero structure computation, see [127].

A similar analysis of stability can be done for the pencil algorithms, see [107]. Accuracy and stability of both Toeplitz and pencil algorithms are then comparable.

Chapitre 4

Null-space of polynomial matrices

Our objective in this chapter is to develop a numerical algorithm to obtain null-spaces of polynomial matrices. Let $A(s)$ be a polynomial matrix of size $m \times n$ and rank r. The right *polynomial null-space* of $A(s)$ is the set of non-zero polynomial vectors $v(s)$ such that

$$A(s)v(s) = 0. \tag{4.1}$$

A basis of the right null-space of $A(s)$ is formed by any set of $n - r$ linearly independent vectors satisfying (4.1). We gather these vectors in the columns of a matrix $V(s)$ such that

$$A(s)V(s) = 0.$$

Let δ_i for $i = 1{:}r$ be the degree of each vector in the basis $V(s)$. If the sum of all the degrees δ_i is minimal, then $V(s)$ is a *minimal basis*, in the sense of Forney [25], of the null-space of $A(s)$. Degrees δ_i in a minimal basis are called here the structural indices of the null-space of $A(s)$ [1]. Analogously, a basis of the left null-space of $A(s)$ is formed by any set of $m-r$ linearly independent non-zero vectors satisfying $A^T(s)v^T(s) = 0$. Because of this duality, in the remainder of this chapter we analyze only the right null-space of $A(s)$.

As well as for the zero structure, computing the null-space of a polynomial matrix has several applications in control theory. Consider a linear system in the form (2.2), then the degrees of the vectors in a minimal basis of the null-space of the pencil

$$P(s) = \begin{bmatrix} sI - A & B \\ C & D \end{bmatrix}$$

correspond to the invariant lists I_2 and I_3 defined in [75]. These invariants are key information when solving problems of structural modifications for linear systems [65, 78]. Computing the null-space is also critical when solving the problem of column reduction of a polynomial matrix [77]. Column reduction is the initial step in several elaborated algorithms. Column reducedness of polynomial matrices is a property often required in CACSD [50]. With the polynomial control theory approach,

1. Also called the Kronecker invariant indices [105].

the solution of several control problems has been reformulated in terms of polynomial matrix equations or Diophantine equations [59]. Consider, for instance, the typical polynomial matrix equation $A(s)X(s) = B(s)$. By analogy with the constant case, the null space of $\begin{bmatrix} A(s) & -B(s) \end{bmatrix}$ contains key information about the existence and the uniqueness of solution $X(s)$. Other operations often required in the polynomial approach are easily solved by computing the null-space of a suitable polynomial matrix. Consider for example a linear multivariable system represented by a left coprime factorization $T(s) = D_L^{-1}(s)N_L(s)$, it can be shown [6, 50] that a right coprime factorization $T(s) = N_R(s)D_R^{-1}(s)$ can be obtained via the null-space computation:

$$\begin{bmatrix} D_L(s) & -N_L(s) \end{bmatrix} \begin{bmatrix} N_R(s) \\ D_R(s) \end{bmatrix} = 0.$$

In fault diagnostics the residual generator problem can be transformed into the problem of finding the null-space of a polynomial matrix [26]. In Chapter 5 we present an algorithm for J-spectral factorization of a polynomial matrix. When the analyzed polynomial matrix is singular, we need to extract its null-space as an intermediate step of the factorization algorithm. The J-spectral factorization appears in the solution of robust, H_2 and H_∞ optimization problems [34, 60]. As a final application, in Example 4.3 we show how it is also possible to obtain the transfer function of a linear system by computing the null-space of a polynomial matrix.

The canonical Smith form is also a classical tool to analyze the null-space of $A(s)$. Notice that the last $n - r$ columns of $V(s)$ in (3.2) form a basis of the right null-space of $A(s)$. Similarly, the last $m - r$ rows of $U(s)$ in (3.2) form a basis of the left null-space of $A(s)$.

Taking $A(s)$ as in (3.1) and $v(s) = v_0 + v_1 s + v_2 s^2 + \cdots + v_\delta s^\delta$, we can see that, arranging terms of the same power of s, polynomial equation (4.1) is equivalent to the constant linear system of equations

$$\begin{bmatrix} A_d & & \\ \vdots & \ddots & \\ A_0 & & A_d \\ & \ddots & \vdots \\ & & A_0 \end{bmatrix} \begin{bmatrix} v_\delta \\ v_{\delta-1} \\ \vdots \\ v_0 \end{bmatrix} = T_{\delta+1}\mathcal{V} = 0. \tag{4.2}$$

Matrix $T_{\delta+1}$ in (4.2) is then in a block Toeplitz form.

Definition 4.1
We define the *null-space structure* of $A(s)$ as the degrees and the associated vectors in a minimal basis $V(s)$ of its null-space, i.e. $A(s)V(s) = 0$. ∎

Example 4.1
Consider the transfer function

$$T(s) = N_r(s)D_r^{-1}(s) \begin{bmatrix} \frac{s^2}{(1-s)^a} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{s^2}{(1-s)^2} & \frac{s}{1-s} & 0 \\ 0 & 0 & 0 & \frac{s}{1-s} \end{bmatrix}$$

with

$$N_r(s) = \begin{bmatrix} s^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & s \end{bmatrix}, \qquad D_r(s) = \begin{bmatrix} (1-s)^a & 0 & 0 & 0 \\ 0 & 1-s & 0 & 0 \\ 0 & -s & 1-s & 0 \\ 0 & 0 & 0 & 1-s \end{bmatrix}.$$

with an integer parameter a. If $a = 1$ we have the same system as in Example 5.1 in [6]. We can obtain a left coprime factorization $T(s) = D_l^{-1}(s)N_l(s)$ from $V(s)$ which is a solution of

$$A(s)V(s) = \begin{bmatrix} N_r^T(s) & -D_r^T(s) \end{bmatrix} \begin{bmatrix} D_l^T(s) \\ N_l^T(s) \end{bmatrix} = 0. \tag{4.3}$$

Matrix $V(s)$ has the following general form

$$V(s) = \begin{bmatrix} 0 & 0 & 0 & 0 & (1-s)^a \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & (1-s)^2 & 0 \\ 0 & 0 & (s-1) & 0 & 0 \\ 0 & 0 & 0 & 0 & s^2 \\ 0 & 0 & 0 & s^2 & 0 \\ 0 & 0 & 0 & (s-s^2) & 0 \\ 0 & 0 & -s & 0 & 0 \end{bmatrix}.$$

Notice that if we vary a, then the degree of $P(s)$ and the degree of the minimal basis $V(s)$ also vary. ∎

4.1 Algorithms

Following our notation of Section 2.3 and Algorithm 2.1, we distinguish our original polynomial problem (OPP) as the problem of finding a minimal basis of the null-space of a matrix $A(s)$ as in (3.1). From (4.2) we recover the corresponding constant equivalent problem (CEP). Notice that knowing the degrees δ_i, $i = 1{:}n-r$, solving block Toeplitz equation (4.2) yields the expected null-space vectors. Now we

analyze how to obtain these degrees. Let matrix T_i be a block Toeplitz matrix with i block columns in the form

$$T_i = \begin{bmatrix} A_d & & & \\ \vdots & A_d & & \\ A_0 & \vdots & \ddots & \\ & A_0 & & A_d \\ & & \ddots & \vdots \\ & & & A_0 \end{bmatrix}$$

associated to the polynomial matrix $A(s)$ with coefficients A_i as in (3.1).

Proposition 4.1
The following structural information of $A(s)$ is obtained by analyzing iteratively block Toeplitz matrices T_i of increasing dimensions. Let $r_0 = n$, $r_1 = \text{rank}(T_1)$ and $r_i = \text{rank}(T_i) - \text{rank}(T_{i-1})$ for $i > 1$. Then

(a) Indices r_i satisfy $r_i \geq r_{i+1}$.
(b) $x_i = r_i - r_{i+1}$ is the number of vectors of degree i in the minimal polynomial basis.
(c) When $r_i = r$ for some $i = w$, we have determined all the degrees of the vectors.
(d) If $r_w = r$ then $r_i = r$ for $i > w$.
(e) Index w is finite.

■

PROOF: First notice that matrix T_2 has the form

$$T_2 = \begin{bmatrix} \boxed{T_1} & 0 \\ 0 & \boxed{T_1} \end{bmatrix},$$

so, $r_2 = \text{rank}(T_2) - \text{rank}(T_1)$ is equal to the number of linearly independent columns in the second block column of T_2 that we call T_2^*. Suppose that $\text{rank}(T_1) = r_1$ and that the column k of T_1 is linearly dependent, so columns k and $k+n$ in T_2 are also linearly dependent. On the other hand, because of the Toeplitz structure, if column j in T_1 is linearly independent, it may happen that column $j+n$ in T_2, namely column j in T_2^*, is linearly dependent. In other words $\text{rank}(T_2) \leq \text{rank}(T_1) + r_1$, i.e. $r_2 \leq r_1$. In general, matrix T_i can be written as

$$T_i = \begin{bmatrix} \boxed{T_{i-1}} & 0 \\ 0 & \boxed{T_{i-1}^*} \end{bmatrix},$$

so, similarly, $r_i = \text{rank}(T_i) - \text{rank}(T_{i-1})$ is equal to the number of linearly independent columns added in the rightmost block column of T_i that we call T_i^*. Using the same reasoning as above, it follows that $r_i \leq r_{i-1}$ which proves (a).

From (4.2), if T_1 has full rank, then $A(s)$ has no vectors of degree 0 in the minimal basis of its null-space. If the column k in T_1 is linearly dependent, so the kth column of $A(s)$ is a combination of degree 0 of the other columns. Namely, we have a column $v_1(s) = v_{10}s^0$ of degree 0 in $V(s)$. As we explain above, column $k+n$ in T_2, i.e. column k in T_2^* is also linearly dependent on the other columns in T_2^*. A linear combination for the $(k+n)$th column of T_2 could be $\left[\begin{array}{c} 0 \\ \hline v_{10} \end{array}\right]$ so we recover a vector $v_2(s) = v_1(s)$ which cannot be in the basis of the null-space. On the other hand, consider that column j, linearly independent in T_1, is dependent in T_2^*. In this case notice that a linear combination of column $j+n$ in T_2 has the form $\left[\begin{array}{c} v_{21} \\ v_{20} \end{array}\right]$, so vector $v_2(s) = v_{20} + v_{21}s$ is a valid vector in the minimal basis $V(s)$. In conclusion, the number x_1 of vectors of degree 1 in a minimal basis is equal to the number of new linearly dependent columns in T_2^*, i.e. $x_1 = n - r_2 - (n - r_1) = r_1 - r_2$. Similarly, column $j+2n$ in T_3 is a linear combination of the form $\left[\begin{array}{c} 0 \\ \hline v_{21} \\ v_{20} \end{array}\right]$, namely, it corresponds to an invalid vector for the minimal basis $V(s)$. On the contrary, if column q is linearly independent in T_2^* but dependent in T_3^*, then we can find a new vector $v_3(s) = v_{30} + v_{31}s + v_{32}s^2$ in the minimal basis. So, the number x_2 of vectors of degree 2 in a minimal basis is equal to $x_2 = n - r_3 - (n - r_2) = r_2 - r_3$. In general, the number x_i of vectors of degree i in a minimal basis is equal to the number of new linearly dependent columns in T_{i+1}^*, i.e. $x_i = r_i - r_{i+1}$ which proves (b).

Now, suppose the last vectors in the minimal basis $V(s)$ have degree $w-1$, so the sum $x_0 + x_1 + \cdots + x_{w-1}$ is equal to $n - r$, the nullity of $A(s)$. Now using (b) to substitute x_i in the last equation, it follows that $r_w = r$ which proves (c). Point (d) is easily derived from (b) and (c): $x_i = 0$ for $i \geq w$, so $r - r_{i+1} = 0$.

Finally, from (3.11) we know that $n_r = \delta_1 + \cdots + \delta_{n-r}$ is finite, so no vector in the minimal basis $V(s)$ can have infinite degree. In other words, $r_w = r$ for a finite index w which proves (e). Notice that $w \leq rd+1$. A tight upper bound for w is given by

$$w \leq \sum_{i=1}^{n} \deg A_i(s) - \min_i \deg A_i(s) + 1 = \delta_{\max},$$

with $\deg A_i(s)$ denoting the degree of the ith column of $A(s)$ [37]. □

Now, based on the last proposition and its proof we can sketch an algorithm for the null-space computation of a polynomial matrix. Features of this algorithm will depend on the rank revealing method (RRM) used to compute the constant null-spaces of the block Toeplitz matrices. As well as for the zero structure algorithms, we can use orthogonal methods and compute the structural indices and the vectors at the same time. We can also use the R echelon form to obtain all the information from

the reduction of a large Toeplitz matrix T_w. The analysis done in Sections 3.3.1 and 3.3.2 is also valid here. We can check that the use of the R echelon form, without an optimal way to determine the growth ratio of the dimension of T_j at each step, could imply an unnecessary computational effort, cf. Example 3.7. We can also check that a blocked formulation as in Algorithm 3.3 has no negative impact in the accuracy of the result, cf. Example 3.8. So, in this chapter we only propose the use of orthogonal methods like the LQ factorization with a blocked formulation. The algorithm reads as follows.

Algorithm 4.1 (`null_orthblk`) *For a given polynomial matrix $A(s)$ as in (3.1) this algorithm computes the null-space structure, i.e. the degrees and the associated vectors of a minimal null-space basis $V(s)$, such that $A(s)V(s) = 0$*[2].

0. Let $\bar{\rho}_0 = \bar{r}_0 = \rho_0 = c = 0$, $\bar{W} = L = [\,]$, $r_0 = n$ and $i = 1$.

i. Compute the LQ factorization

$$TQ = \left[\begin{array}{c|c} L(m+1:(d+1)m, \bar{r}_{i-1}+1:(i-1)n) & T_1 \\ 0 & \end{array}\right] Q = L.$$

Compute the rank r_T of T and count the number $\bar{r}_i$ of linearly independent rows in its first block row. Let $N = Q(:, \bar{r}_i + 1 : in - \bar{\rho}_{i-1})$ and $\bar{\rho}_i = \bar{\rho}_{i-1} + \bar{r}_i$. Let $\rho_i = r_T + \bar{\rho}_{i-1}$ and $r_i = \rho_i - \rho_{i-1}$. Let

$$\bar{W} = \left[\begin{array}{cc} \bar{W} & 0 \\ 0 & I_n \end{array}\right] N, \qquad W = \bar{W}(:, r_T - \bar{r}_i + 1 : in - \bar{\rho}_i),$$

and $x_{i-1} = r_{i-1} - r_i$. From W, choose x_{i-1} columns and arrange them in a matrix X. Now let $\mathcal{V}_{j+c} = X(:, j)$ and $\delta_{j+c} = i - 1$ for $j = 1{:}x_{i-1}$ be the vectors and the degrees in a minimal basis.
If $r_i = r$, then go to step f1.
Finally let $c = c + x_{i-1}$ and $i = i + 1$.

f1. The vectors $v_i(s)$, $i = 1{:}n{-}r$ which forms a minimal basis $V(s) = [v_1(s) \cdots v_{n-r}(s)]$ are given by

$$v_i(s) = \mathcal{V}_i(1:n)s^{\delta_i} + \mathcal{V}_i(n+1:2n)s^{\delta_i - 1} + \cdots + \mathcal{V}_i(\delta_i n + 1 : (\delta_i + 1)n)$$

Here we also want to investigate the use of displacement structure methods, like the generalized Schur algorithm (GSM), to compute the LQ factorizations at each step. So, we are interested in the fast LQ factorization of $mk \times nl$ block Toeplitz matrices of the form

$$\left[\begin{array}{ccc|ccc} M_0 & \cdots & M_{f-1} & M_f & \cdots & M_{l-1} \\ \vdots & \ddots & \vdots & \vdots & & \vdots \\ M_{-f+1} & \cdots & M_0 & M_1 & \cdots & M_{l-f} \\ \hline M_{-f} & \cdots & M_{-1} & M_0 & \cdots & M_{l-f-1} \\ \vdots & & \vdots & \vdots & & \vdots \\ M_{-k+1} & \cdots & M_{-k+f} & M_{-k+f+1} & \cdots & M_{-k+l} \end{array}\right] \tag{4.4}$$

2. A similar Toeplitz algorithm using the SVD and without a blocked formulation was presented in [5].

with blocks M_i of dimension $m \times n$ and where $f = \min(k, l)$. Notice that a block version of the displacement structure theory and the GSM presented in Section 2.3.4 is required. Some results on the application of the GSM to block Toeplitz matrices are presented in [56, 57], see Section 2.5 in [56] for algorithms to obtain proper block generators for example. A direct application of this *block GSM* allows to compute a *fast LQ factorization* of a matrix T in the form (4.4). We sketch the method as follows, see also [126].

Lemma 4.1
Let matrix $T = LQ$ be as in (4.4). Build the extended matrix

$$\bar{T} = \begin{bmatrix} TT^T & T \\ T^T & I \end{bmatrix}$$

that is symmetric positive semidefinite, and satisfies the block displacement equation

$$\nabla \bar{T} = \bar{T} - F\bar{T}F^T = X\Sigma X^T \tag{4.5}$$

where $F = \text{diag}\{F_m, F_n\}$ with

$$F_m = \begin{bmatrix} 0 & & & \\ I_m & 0 & & \\ & \ddots & \ddots & \\ & & I_m & 0 \end{bmatrix} \in \mathbb{R}^{mk \times mk}, \qquad F_n = \begin{bmatrix} 0 & & & \\ I_n & 0 & & \\ & \ddots & \ddots & \\ & & I_n & 0 \end{bmatrix} \in \mathbb{R}^{nl \times nl}.$$

Compute the LQ factorization $[M_0 \cdots M_{l-1}] = L_0 Q_0$ of the first block row of T, and build $[M_0 \cdots M_{l-1}] = \bar{L}_0 C$ where $\bar{L}_0$ has full column rank $\bar{m} \leq m$ and $C = [C_0 \cdots C_{l-1}]$ contains the first $\bar{m}$ rows of Q_0. Define

$$\begin{bmatrix} Y_0 \\ \vdots \\ Y_{k-1} \end{bmatrix} = T \begin{bmatrix} C_0^T \\ \vdots \\ C_{l-1}^T \end{bmatrix}.$$

Then the generator X of $\bar{T}$ satisfying displacement equation (4.5) with

$$\Sigma = \text{diag}\{I_{\bar{m}+n}, -I_{\bar{m}+n}\}$$

is given by

$$X = [X_1^T \quad X_2^T]^T \tag{4.6}$$

where

$$X_1 = \begin{bmatrix} Y_0 & 0 & 0 & 0 \\ Y_1 & M_{-1} & Y_1 & M_{l-1} \\ Y_2 & M_{-2} & Y_2 & M_{l-2} \\ \vdots & \vdots & \vdots & \vdots \\ Y_{k-1} & T_{-k+1} & Y_{k-1} & T_{-k+l+1} \end{bmatrix}, \quad X_2 = \begin{bmatrix} C_0^T & I_n & C_0^T & 0 \\ C_1^T & 0 & C_1^T & 0 \\ \vdots & \vdots & \vdots & \vdots \\ C_{l-1}^T & 0 & C_{l-1}^T & 0 \end{bmatrix}. \tag{4.7}$$

Finally, by applying the block GSM with generator X we obtain $\bar{T} = WW^T$, and it is easy to see that

$$W = \begin{bmatrix} L \\ Q^T \end{bmatrix}.$$

Moreover, we can check that if w_{ij} is the leading entry of the jth column of L, then the row T_{i*} is linearly independent from the previous rows. ∎

PROOF: See Theorem 2.20 in [56] and Example 5.3 in [51]. □

Matrices T_i in Algorithm 4.1 are a particular case of matrix (4.4) where $k = d+i$, $l = i$ and where m and n are the dimensions of the polynomial matrix $A(s)$. So we can reduce matrix T_i by applying the fast LQ factorization described above. Because of the particular form of T_i, its generator matrix can be obtained from the generator of T_{i-1} as follows. Consider the LQ factorization

$$A_d = [\Delta \quad 0] \begin{bmatrix} M^T \\ N^T \end{bmatrix}.$$

A generator of T_1 is then given by $G_1 = [X_1^T \quad Y_1^T]^T$ where

$$X_1 = \begin{bmatrix} \boxed{\begin{matrix} \Delta & 0 & 0 \\ A_{d-1}M & A_{d-1} & A_{d-1}M \\ A_{d-2}M & A_{d-2} & A_{d-2}M \\ \vdots & \vdots & \vdots \\ A_0M & A_0 & A_0M \end{matrix}} & \boxed{\begin{matrix} 0 \\ A_d \\ A_{d-1} \\ \vdots \\ A_1 \end{matrix}} \end{bmatrix}, \quad Y_1 = \begin{bmatrix} M & I_n & M & 0 \end{bmatrix}. \tag{4.8}$$

At the next step, the LQ factorization of the first block row of T_2 is given by

$$[A_d \quad 0] = [\Delta \quad 0 \quad 0] \left[\begin{array}{cc} M^T & 0 \\ N^T & 0 \\ \hline 0 & \times \end{array}\right],$$

so the corresponding generator is given by $G_2 = [X_2^T \quad Y_2^T]^T$ where

$$X_2 = \begin{bmatrix} \boxed{\begin{matrix} \Delta & 0 & 0 \\ A_{d-1}M & A_{d-1} & A_{d-1}M \\ A_{d-2}M & A_{d-2} & A_{d-2}M \\ \vdots & \vdots & \vdots \\ A_0M & A_0 & A_0M \end{matrix}} & \boxed{\begin{matrix} 0 \\ 0 \\ A_d \\ A_{d-1} \\ \vdots \\ A_1 \end{matrix}} \\ 0 & \end{bmatrix},$$

$$Y_2 = \begin{bmatrix} M & I_n & M & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

In summary, if the fast LQ factorization is used at each step in Algorithm 4.1, then some of the information obtained at step i can also be used in the computations at step $i+1$, i.e. a blocked formulation is possible. The algorithm reads as follows.

Algorithm 4.2 (`null_gsm`) *For a given polynomial matrix $A(s)$ as in* (3.1) *this algorithm computes the null-space structure.*

1. *Compute $G_1 = [X_1^T \quad Y_1^T]^T$ as in* (4.8) *and partition $X_1 = [X_{1_1} \quad X_{1_2}]$ where X_{1_2} contains the last n columns of X_1. Apply the GSM to obtain the factorization $T_1 Q = L$. Let $r_1 = \rho_1 = \mathrm{rank}(T_1)$. Let $x_0 = n - r_1$. From $W = Q(:, \rho_1 + 1 : n)$ choose x_0 columns and arrange them in a matrix X. Now let $\mathcal{V}_j = X(:, j)$ and $\delta_j = 0$ for $j = 1{:}x_0$ be the vectors and the degrees in a minimal basis.*
   If $r_1 = r$, then go to step s1.
   Finally let $c = x_0$ and $i = 2$.

i. *Build the proper generator $G_i = [X_i^T \; Y_i^T]^T$ where*

$$X_i = \left[\begin{array}{c|c} X_{(i-1)_1} & \begin{array}{c} 0 \\ \\ X_{(i-1)_2} \end{array} \\ \hline 0 & \end{array}\right], \quad Y_i = \left[\begin{array}{c} Y_{i-1} \\ 0 \end{array}\right].$$

   Apply one step of the GSM, namely, displace the first block column of G_i and process the first m rows of the displaced generator in order to obtain a new proper generator $\bar{G}_i = [\bar{X}_i^T \quad \bar{Y}_i^T]^T$. Partition $\bar{X}_i = [X_{i_1} \quad X_{i_2}]$ where X_{i_2} contains the last n columns of $\bar{X}_i$. Apply the GSM with generator $\bar{G}_i$ to obtain the LQ factorization $T_i Q = L$. Let $\rho_i = \mathrm{rank}(T_i)$ and $r_i = \rho_i - \rho_{i-1}$. Let $x_{i-1} = r_{i-1} - r_i$. From $W = Q(:, \rho_i + 1 : n)$ choose x_{i-1} columns and arrange them in a matrix X. Now let $\mathcal{V}_{j+c} = X(:, j)$ and $\delta_{j+c} = i - 1$ for $j = 1{:}x_{i-1}$ be the vectors and the degrees in a minimal basis.
   If $r_i = r$, then go to step s1.
   Finally let $c = c + x_{i-1}$ and $i = i + 1$.

s1. *See step f1 of* Algorithm 4.1.

4.2 Algorithmic analysis

In this section we analyse the performance of the different algorithms presented above. Another objective of this section is to compare our algorithms with the standard algorithms in the literature to compute null-spaces of polynomial matrices. These standard algorithms are based on the linearization of the polynomial matrix $A(s)$ (the pencil approach, cf. Section 2.3 and [6]), and can be sketched as follows.

Algorithm 4.3 (`null_pencil`)

1. *A pencil $sB_0 - A_0$ associated to the linearization of $A(s)$ is transformed in the generalized Schur form*

$$U(sB_0 - A_0)V = \left[\begin{array}{cc} sB_z - A_z & * \\ 0 & sB_* - A_* \end{array}\right] \tag{4.9}$$

   by the methods explained in [105] *and sketched at the beginning of* Section 3.3 *with* Algorithm 3.6.

2. *A minimal basis* $Z_z(s)$ *of* $sB_z - A_z$*, which is in upper staircase form and contains only the infinite and null-space structural indices of* $A(s)$*, is computed with the recursive process presented in* Section 4 *of* [6].
3. *Finally the* n *bottom rows of* $V[\ Z_z^T(s)\ \ 0\ ]^T$ *are the required basis vectors.*

4.2.1 Complexity

As we noted in Section 3.3.1. the pencil Algorithm 3.6 to obtain infinite structural indices computes also the null-space structural indices. Notice also that Algorithms 3.3 and 4.1 are very similar, and the complexity of Algorithm 4.1 is $O(dn^3)$ [126]. In that way the conclusions given in Section 3.3.1 are also valid here. One advantage of the Toeplitz approach is the weak dependence on the matrix degree when the null-space structure is fixed. In the general case, the type of matrices for which the pencil Algorithm 3.6 is more efficient than the Toeplitz Algorithm 4.1 are matrices with a small degree d and a small nullity but with large null-space structural indices. However, we can also check that the number of flops performed by both algorithms in these cases will be comparatively large. On the other hand, we can easily find a polynomial matrix very favourable for the Toeplitz algorithm and simultaneously very unfavourable for the pencil algorithm. But the major advantage of the Toeplitz Algorithm 4.1 is that not only the structural indices but also the null-space vectors are obtained with the same computational effort. On the contrary, the algorithm in Section 4 of [6], used at step 2 in Algorithm 4.3, typically requires a large number of operations, which have to be added to the amount of operations performed by the first part of the algorithm, i.e. the computation of Schur form (4.9). We illustrate this with the following examples. To compare the performance of the algorithms we compare their execution times. Algorithm 3.6 was programmed also using the LQ factorization.

Example 4.2
Consider the problem of Example 4.1. In Table 4.1 we present the execution times in seconds needed by Algorithm 4.1 and the pencil Algorithm 4.3 to solve equation (4.3) for different values of a. The time of the pencil algorithm is split in two: first the time required to compute the pencil (4.9) using Algorithm 3.6, and second the time required to compute the null-space of $sB_z - A_z$ with the algorithm of Section 4 in [6][3]. Notice the fast growth of the execution time of the pencil algorithm with respect to the Toeplitz algorithm. When the degree of $V(s)$ grows, the number of steps also grows and therefore the number of columns of the last analyzed Toeplitz matrices can be large. Nevertheless, because of the blocked formulation of Algorithm 4.1, the actual number of columns analyzed at step i is smaller than in, which has a positive impact on the execution time. ∎

3. This algorithm was programmed with the functions of the Polynomial Toolbox, so the execution time reported here could be reduced.

a	`null_pencil`	`null_orthblk`
3	0.07 + 0.69	0.11
5	0.14 + 1.12	0.15
10	0.52 + 2.76	0.37
15	1.4 + 4.52	0.72
20	4.13 + 8.23	1.46

TAB. 4.1 – *Execution times in seconds of the different algorithms when solving equation* (4.3) *for different values of* a.

Example 4.3
Now consider the interconnected mass-spring system of Figure 4.1. This system is governed by a second-order linear differential equation of the type

$$M\frac{\mathrm{d}^2}{\mathrm{d}t^2}x + Kx = Bu, \tag{4.10}$$

where matrices M and K contains the values m_i of the masses and k_i of the spring constants, for $i = 1{:}p$ with p the state dimension. Let $m_i = k_i = 1$ for $i = 1{:}p$. We want to compute the transfer function matrix from the input u to the output y, the position of the last mass, i.e. $y = Cx = [0\ 0\ \cdots\ 1]x$. The Laplace transform of equation (4.10) is given by $D(s)x = Bu$, where

$$D(s) = Ms^2 + K = \begin{bmatrix} 1+s^2 & -1 & & & \\ -1 & 2+s^2 & -1 & & \\ & -1 & 2+s^2 & & \\ & & & \ddots & -1 \\ & & & -1 & 2+s^2 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

The system transfer function is given by

$$G(s) = CD^{-1}(s)B = C\frac{N(s)}{d(s)} = \frac{n(s)}{d(s)},$$

from which it follows that $D(s)N(s) = Bd(s)$. So, from the vector of minimal degree of the null-space of $A(s) = [\ D(s)\ \ -B\ ]$, we can obtain the numerator $n(s)$ and the denominator $d(s)$ of the transfer function as follows:

$$[\ D(s)\ \ -B\ ]\,[\ \times\ \ \times\ \ \cdots\ \ n(s)\ \ d(s)\ ]^T = 0. \tag{4.11}$$

Matrix $D(s)$ has dimension $p \times p$, so polynomial matrix $A(s) = [\ D(s)\ \ -B\ ]$ has full row rank p, no finite zeros and, since $\operatorname{rank} A_2 = p$, it has no infinite zeros either. So, from (3.11), the degree of the unique vector in the minimal basis of the null-space of $A(s)$ is always $\delta_{\max} = 2p$. In Table 4.2 we present the execution times required by the pencil and Toeplitz algorithms to compute the null-space of $A(s)$. Now notice that the execution time of the Toeplitz algorithms grows significantly. In fact, when p is large, matrix $A(s)$ represents the worst case for Algorithm 4.1: it is necessary

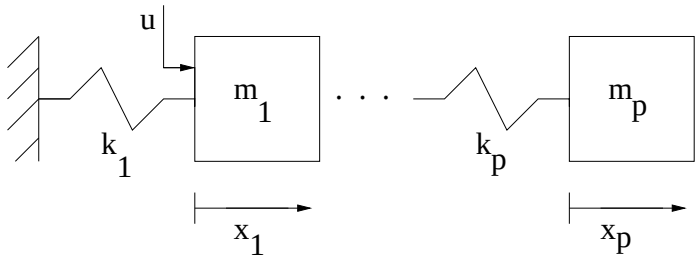

FIG. 4.1 – *Connected mass-spring system with p masses.*

p	`null_pencil`	`null_orthblk`	`null_gsm`
3	0.11 + 1.64	0.11	0.12
5	0.18 + 3.06	0.20	0.38
10	0.42 + 7.14	0.56	1.20
15	0.95 + 14.57	1.27	4.94
20	1.80 + 24.79	3.84	7.83

TAB. 4.2 – *Execution times in seconds for the different algorithms when solving equation* (4.11) *of increasing dimensions p.*

to perform the maximum number of steps and the analyzed Toeplitz matrices are large. On the other hand the degree of $A(s)$ is always 2 and all the SVD performed to obtain the generalized form (4.9) have dimensions smaller than $2p \times (2p+1)$. This has a positive impact on the execution time of the first part of the pencil Algorithm 4.3. The problem is that at this step we have not yet the vector which forms the minimal basis of the null-space of $A(s)$ but only its degree. If we want to obtain the vector, then a large execution time is required by the procedure outlined in [6]. With this example we can see the advantage of the Toeplitz algorithms: we obtain, with the same computational effort, the degrees and the vectors of the minimal null-space basis. ∎

Now we investigate how Algorithm 4.2, which uses the fast LQ factorization, performs with respect to Algorithm 4.1. First we analyze the performance of the fast LQ factorization applied on a non-square block Toeplitz matrix T as in (4.4). As we noted in Seection 2.3.4, processing the few columns of the generator matrix allows to expect a gain of one order of magnitude in the complexity expression of the factorization. Unfortunately when the analyzed matrix is block Toeplitz, this better performance is not always guaranteed. The fast LQ factorization of T achieves its maximal performance, in comparison with the standard factorization, when processing a generator with a number of columns smaller than the number of columns of T. The number of columns of matrix T grows with the number of columns of its blocks or with the number of block columns. On the other hand, the number of columns of the generator given by (4.6) and (4.7) decreases if the rank of the first block row of T is small. The rank of the first block row of T is less than or equal to $\min(m, nl)$. In conclusion, in order to ensure the best performance of the fast factorization we have

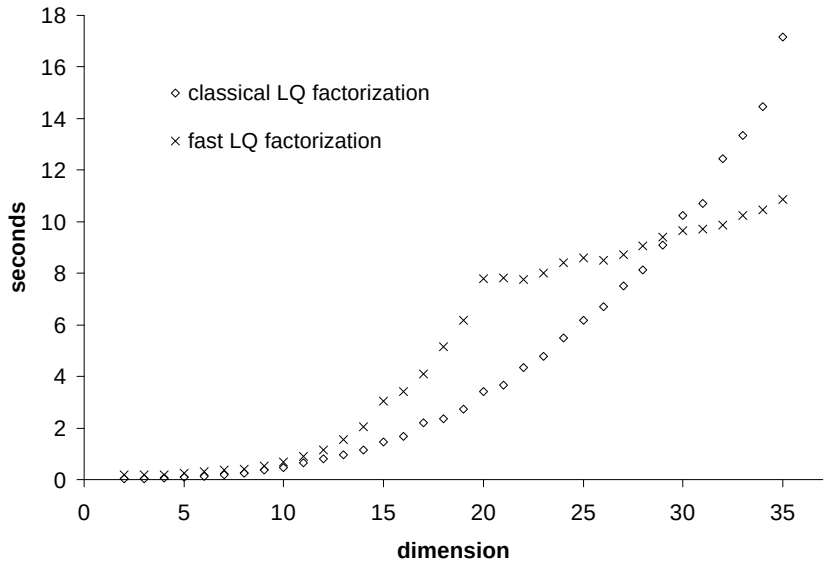

FIG. 4.2 – *Execution times for different numbers of columns in the blocks of a block Toeplitz matrix (*4.4*) with 7 block columns.*

to apply it onto block Toeplitz matrices in form (4.4) with $n \geq m$ moderately large and with l large. Notice that the number of block rows k has the same influence in both standard and fast factorizations.

In Figure 4.2 we present the execution times of the standard LQ and the fast LQ factorization of a block Toeplitz matrix in form (4.4) with $m = 20$, $k = 10$, $l = 7$ and $n = 2{:}35$. Notice that the fast factorization achieves its best performance when $n \geq 20$. The value of n from which the fast factorization begins to be more efficient than the standard one is different for each matrix T and seems to be difficult to predict. Nevertheless we can see that if l is larger, then the fast factorization is more efficient for smaller values of n. If we take the same matrix used in Figure 4.2 but with $l = 10$, we can see that the fast factorization is more efficient than the standard one even when $m \geq n$.

Now we analyze the performance of Algorithm 4.2. Consider an $m \times n$ polynomial matrix $A(s)$ with degree d. The total amount of flops performed by the algorithm is equal to the sum of flops performed at each step. Therefore, the final execution time is the sum of the execution times needed to obtain the null-spaces of the different Toeplitz matrices T_i. In Figure 4.3 we present the execution times of the standard and the fast LQ factorization applied onto a block Toeplitz matrix in form (4.4) with $m = 18$, $n = 18$, $k = 2 + l$, and $l = 3{:}17$. This corresponds to the application of Algorithms 4.1 and 4.2 onto a 18×18 polynomial matrix of degree 2 which has at least one vector of degree 16 in the basis of its null-space. Notice that even if the standard LQ factorization is more efficient when $l \leq 10$, the total execution time of Algorithm 4.1 is larger than the total execution time of Algorithm 4.2. Graphically

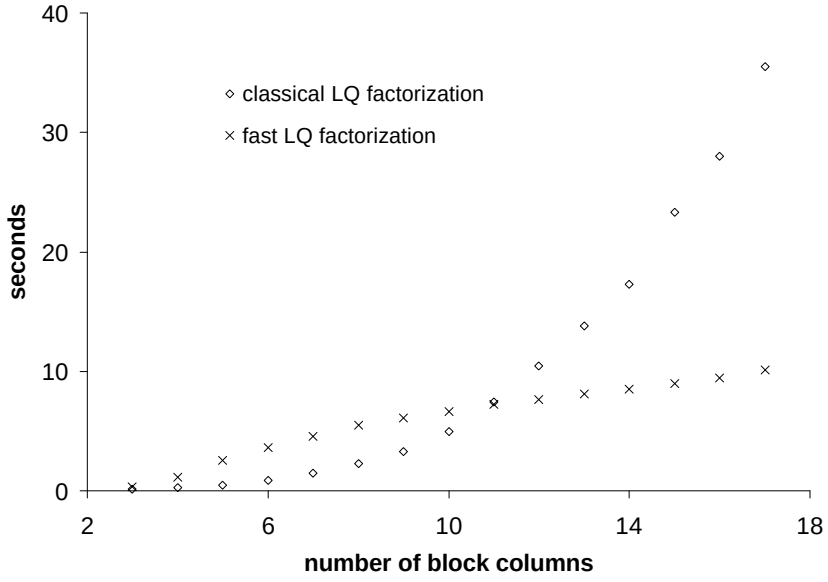

FIG. 4.3 – *Execution times for different numbers of block columns in a block Toeplitz matrix (*4.4*).*

we can consider that the total execution times are the surfaces below the curves in Figure 4.3. Obviously we can also find polynomial matrices for which Algorithm 4.1 is more efficient than its fast version. Consider for example a 18×18 polynomial matrix of degree 2 and consider that it has a null-space of maximal degree equal to 6. From Figure 4.3 we can see that the sum of the execution times for $l = 3, 4, 5, 6, 7$ is larger for the fast factorization than for the standard factorization.

Example 4.4
Consider the mass-spring system of Example 4.3. The execution times required by Algorithm 4.2 to solve equation (4.11) for different number of masses p, are also presented in Table 4.2. Notice that in this case the fast LQ factorization does not achieve its best performance and thus, Algorithm 4.1 is more efficient even when p is large. For Algorithm 4.2 to be more efficient it needs to perform more steps. In other words, for a same number of masses, a polynomial matrix $[D(s) \ -B]$ having a null-space of degree greater than $2p$ would be needed. Nevertheless, from equation (3.11) we can see that this is not possible. In conclusion, Algorithm 4.2 should be applied carefully when a small number of executed flops is required. ■

4.2.2 Stability and accuracy

The analysis and conclusions in Section 3.3.2 of this thesis are also valid here. Lemma 3.4 is reformulated as follows.

Lemma 4.2
Consider that polynomial matrix $A(s)$ has a vector of degree δ in a basis of its null-space. The application of the blocked LQ factorization (as in Algorithm 4.1) to solve system (4.2) is backward stable, namely

$$(T_{\delta+1} + \Delta)\hat{\mathcal{V}} = 0, \qquad \|\Delta\|_2 \leq \phi\|T_{\delta+1}\|_2.$$

where $\hat{\mathcal{V}}$ contains the computed coefficients of the vectors in the null-space basis and ϕ is a small constant depending on the dimensions of $T_{\delta+1}$, the machine precision ε, and the number of applied Householder reflections. ∎

This result shows that $\hat{\mathcal{V}}$ is the exact null-space of the slightly perturbed matrix $T_{\delta+1} + \Delta$. Nevertheless, what we want is that vector $\hat{v}(s)$, constructed from $\hat{\mathcal{V}}$, is the exact polynomial vector in the null-space of the slightly perturbed matrix $A(s)+\Delta(s)$ where $\Delta(s) = \Delta_0+\Delta_1+\cdots+\Delta_d s^d$ is small for some polynomial matrix norm. Ideally we want

$$\|\Delta_i\|_2 \leq \gamma\|A_i\|_2, \qquad i = 0, 1, \ldots, d.$$

Equivalently, we want that $(T_{\delta+1} + \Delta_T)\hat{\mathcal{V}} = 0$ where

$$\Delta_T = \begin{bmatrix} \Delta_d & & \\ \vdots & \ddots & \\ \Delta_0 & & \Delta_d \\ & \ddots & \vdots \\ & & \Delta_0 \end{bmatrix}, \qquad \|\Delta_i\| \leq \gamma\|A_i\|_2 \quad i = 0, 1, \ldots, d$$

and where $\hat{\mathcal{V}}$ is the computed null-space of $T_{\delta+1}$. The following result gives a bound for the error $\Delta(s)$ when using the LQ factorization to solve (4.2), see also [127].

Theorem 4.1
Let $A(s)$ be as in (3.1) and suppose that it has a vector $v(s)$ of degree δ in a basis of its null-space. The computed vector $\hat{v}(s)$, obtained from (4.2) via the blocked LQ factorization, is the exact null-space vector of the slightly perturbed matrix $A(s) + \Delta(s)$ with

$$\|\Delta_i\|_2 \leq \gamma\|T_{\delta+1}\|_2, \qquad i = 0, 1, \ldots, d. \tag{4.12}$$

∎

PROOF: From Lemma 4.2 we know that $(T_{\delta+1}+\Delta)\hat{\mathcal{V}} = 0$ with $\|\Delta\|_2 \leq \gamma\|T_{\delta+1}\|_2$. Error matrix Δ has the following general form:

$$\Delta = \begin{bmatrix} \Delta_d^0 & & \\ \vdots & \ddots & \\ \Delta_0^0 & & \Delta_d^\delta \\ & \ddots & \vdots \\ \times & & \Delta_0^\delta \end{bmatrix}$$

where $\times$ represents possibly non zero blocks. Nevertheless, we can always apply row elementary operations gathered in a left multiplier $P = I + E$ with $\|E\|_2 \leq \gamma \|T_{\delta+1}\|_2$ such that

$$P(T_{\delta+1} + \Delta)\hat{\mathcal{V}} = (T_{\delta+1} + \Delta_T)\hat{\mathcal{V}} = 0.$$

Transformation matrix P and thus Δ_T are not unique but we can check that $\|\Delta_T\|_2 \leq \gamma \|T_{\delta+1}\|_2$ or equivalently $\|\Delta_i\|_2 \leq \gamma \|T_{\delta+1}\|_2$ which is the expected result. □

Now, in order to estimate the value of γ in (4.12) we simply consider the standard definition of relative backward error as in [44], so we look for the smallest value satisfying (4.12). Namely,

$$\gamma = \min\{\epsilon : (A(s) + \Delta(s))\hat{v}(s) = 0, \quad \|\Delta(s)\|_2 = \|\Delta_T\|_2 \leq \epsilon \|T_{\delta+1}\|_2\}. \tag{4.13}$$

Taking the norm of the residual $r(s) = A(s)\hat{v}(s) = -\Delta(s)\hat{v}(s)$ we obtain a lower bound for ϵ

$$\|r(s)\|_2 \leq |\epsilon| \|T_{\delta+1}\|_2 \|\hat{v}(s)\|_2,$$

so, the solution of (4.13) is

$$\gamma = \frac{\|r(s)\|_2}{\|T_{\delta+1}\|_2 \|\hat{v}(s)\|_2}. \tag{4.14}$$

Sharper bounds than (4.12,4.14) can be expected from a componentwise error analysis [44, 120]. An analysis with geometrical arguments such as in [23, 119] can also be considered as a future line of research.

Although backward stability of the GSM can be guaranteed, cf. Section 2.3.4, when the method is extended to block Toeplitz matrices this property is lost [52]. Similarly, stability of the process in Section 4 of [6], used at step 2 in Algorithm 4.3 to obtain the vectors in a minimal basis, is not guaranteed. So, backward stability of the pencil Algorithm 4.3 and Algorithm 4.2 is not guaranteed. This implies an additional advantage of our Algorithm 4.1.

4.2.3 Extension of the Toeplitz algorithms

We have seen how the Toeplitz algorithms are reliable alternatives to the pencil algorithms. Toeplitz algorithms are in general faster than pencil algorithms and the accuracy of the obtained results is at least as good as for the pencil algorithms. One advantage of the pencil algorithm is that we can extract from $sB_z - A_z$ in (4.9) the structural indices of the null-space of $A(s)$ and also its infinite structural indices. Moreover, we do not require the rank of $A(s)$ since we can also deduce it from $sB_z - A_z$. In this section we show how the Toeplitz algorithm allows to obtain not only $V(s)$ but also the infinite structural indices, the associated infinite vectors and the rank of $A(s)$ with the same computational effort [124].

Notice that Toeplitz matrix in (3.14) is equal to the uppermost rows of the Toeplitz matrix in (4.2), so, indices $\bar{r}_i$ and $\bar{\rho}_i$ obtained by Algorithm 4.1 contain the infinite structure of $A(s)$, cf. Algorithm 3.3. The problem is that the rank r of $A(s)$ is required so as to know when Algorithms 3.2 or 4.1 have to stop. When r is not

known however, we can simply start with the guest $r = \min(m, n)$ and update this value as the vectors in the minimal basis $V(s)$ are found. We illustrate the process with the following simple example.

Example 4.5
We want to obtain the infinite and the null-space structure of matrix

$$A(s) = \begin{bmatrix} 1 & s^3 & 0 & 0 \\ 0 & 1 & s & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Supposing that we do not know the rank of $A(s)$, we can start with the guess $r = \min(m, n) = 3$. After the first step of Algorithm 4.1 we obtain $\bar{r}_1 = 1$ and $r_1 = 3$ which means that there is one vector of degree 0 in $V(s)$. At this moment Algorithm 4.1 could stop since $r_1 = r$, nevertheless $\bar{r}_1 \neq r$ which means that there are zeros at infinity and therefore that the rank of $A(s)$ is not 3. After other 3 steps of Algorithm 4.1 we obtain values

$$\begin{aligned} \bar{r}_2 &= 1, \quad r_2 = 3, \\ \bar{r}_3 &= 2, \quad r_3 = 3, \\ \bar{r}_4 &= 2, \quad r_4 = 3, \end{aligned}$$

which confirms the existence of zeros at infinity in $A(s)$. Finally, at the next step we obtain

$$\bar{r}_5 = 2, \qquad r_5 = 2.$$

At this moment we know that r is at most equal to 2 because we have $r_4 - r_5 = 1$ vector of degree 4 in the minimal basis $V(s)$. Moreover, we notice that $\bar{r}_3 = r_5 = r$ which means that all the infinite eigenvectors and the null-space vectors have been obtained. The algorithm stops. We conclude that the rank 2 matrix $A(s)$ has one vector of degree 0 and one vector of degree 4 in a minimal basis of its null-space, and also $\bar{r}_3 - \bar{r}_2 = 1$ chain of eigenvectors at infinity of length 2, i.e. 2 zeros at infinity. Notice that equation (3.11) is verified. ∎

Chapitre 5

Polynomial J-spectral factorization

Our objective in this chapter is to develop numerical algorithms to factorize polynomial matrices. As for the constant case, there can be different types of factorizations of a polynomial matrix $A(s)$. In Chapter 3 we have seen two useful factorizations: the Smith form (3.2) and the Smith MacMillan form at infinity (3.7), resulting in diagonal factorizations of $A(s)$.

Here we are interested in the factorization of $A(s)$ in polynomial factors containing a desired part of its eigenstructure. Our final objective is the polynomial J-spectral factorization, an important tool of analysis and design of control systems, cf. Section 5.2.

5.1 Factorization of the eigenstructure

Factor extraction on a polynomial matrix $A(s)$ consists in finding a right factor $R(s)$ containing a desired part of its eigenstructure, for instance a set of finite zeros with their respective chains of eigenvectors, and such that

$$A(s) = L(s)R(s).$$

Left factor $L(s)$ contains the remainder of the eigenstructure of $A(s)$. It is not always possible to extract in $R(s)$ any arbitrary part of the eigenstructure of $A(s)$. Theoretical conditions are presented in Section 7.7 of [31]. Here, for the effects of our algorithms, we rather analyze this problem in a practical sense using Lemma 3.3 and Corollary 3.1.

Consider a square full-rank polynomial matrix $A(s)$ of dimension n and degree d with a set $\{z_1, z_2, \ldots, z_k\}$ of finite zeros and with m_∞ zeros at infinity [1]. From (3.11)

1. Extension to the non-square case is direct.

it follows that $nd = k + m_\infty$. Suppose we want to extract an $n \times n$ factor $R(s)$ of degree d_R containing a subset of $\bar{k}$ finite zeros of $A(s)$:

- If $\bar{k} = nd_R$ and $R(s)$ contains only the $\bar{k}$ finite zeros of $A(s)$, then we have an *exact factorization*.
- If $\bar{k}$ is not an integer multiple of n, then the exact factorization of a subset of $\bar{k}$ finite zeros of $A(s)$ is not possible. In this case we can show that $R(s)$ contains the $\bar{k}$ finite zeros but also some zeros at infinity. The factorization is then inexact.

The condition that the degree $d_R = \bar{k}/n$ of factor $R(s)$ should be an integer is only a necessary condition to have an exact factorization. We also require that equation $L(s)R(s) = A(s)$ can be solved for a polynomial factor $L(s)$. Solvability of this polynomial matrix equation is related to the fact that $A(s)$ and the extracted factor $R(s)$ have the same Jordan pairs associated to the extracted zeros as explained in Chapter 7 of [31].

Example 5.1
Consider the matrix

$$A(s) = \begin{bmatrix} s & -s^2 \\ 1 & 0 \end{bmatrix}$$

which has two finite zeros at $s = 0$ and two zeros at infinity. Suppose we want to extract exactly a factor $R(s)$ containing the two finite zeros. We can see that the degree of $R(s)$ should be $d_R = 2/2 = 1$. For instance we can propose

$$R(s) = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix}.$$

Nevertheless there is no solution $L(s)$ for the polynomial equation $L(s)R(s) = A(s)$. As a result, $R(s)$ is not a valid right factor of $A(s)$. One possible factorization is given by

$$R(s) = \begin{bmatrix} 1 & 0 \\ 0 & s^2 \end{bmatrix}, \quad L(s) = \begin{bmatrix} s & -1 \\ 1 & 0 \end{bmatrix}.$$

Notice that $R(s)$ has the two finite zeros at $s = 0$ but also two zeros at infinity, so the factorization is inexact. ∎

By analogy, we say that $A(s)$ can have an exact factorization of its infinite zeros whenever m_∞ is an integer multiple of n. By duality, notice that an inexact factorizations of infinite zeros introduce zeros at $s = 0$, cf. Example 5.4. When $A(s)$ has rank $r < n$, from (3.11) we can see that for $A(s)$ to have an exact factorization of its right null-space, $R(s)$ should have full row-rank equal to r and its degree d_R should satisfy $rd_R = n_r$. When the factorization of the null-space is not exact, $R(s)$ has also some zeros at infinity, see Examples 5.3 and 5.5.

5.1.1 Extraction of a set of zeros

Here we present an algorithm to extract a valid factor $R(s)$ containing a desired set $z = \{z_1, z_2, \ldots, z_q\}$ (including maybe infinity) of zeros of a square non-singular $n \times n$ polynomial matrix $A(s)$. The zero z_j for $j = 1, 2, \ldots, q$ has a number m_G of chains of associated eigenvectors equal to its geometric multiplicity. The ith chain of eigenvectors satisfies (3.5) or (3.10). Define

$$V_i = \begin{bmatrix} v_{i1} & v_{i2} & \cdots & v_{ik_i} \\ & v_{i1} & \cdots & v_{i(k_i-1)} \\ & & \ddots & \vdots \\ 0 & & & v_{i1} \end{bmatrix}$$

for $i = 1, 2, \ldots, m_G$ and build up the matrix $W_j = [V_1 \cdots V_{m_G}]$ called the characteristic matrix of z_j. If $R(s)$ is a factor containing the zero z_j, then it can also be associated the same eigenvectors. In other words, it can be shown [4, 39] that $R(s) = R_0 + R_1 s + \cdots + R_d s^d$ is obtained from the basis of the left null-space of $[W_1 \cdots W_q]$, namely,

$$\begin{bmatrix} R_0 & R_1 & \cdots & R_d \end{bmatrix} \begin{bmatrix} W_1 & W_2 & \cdots & W_q \end{bmatrix} = RW = 0. \tag{5.1}$$

To solve this last equation we can use some of the numerical linear algebra methods described in Section 2.3. Notice that matrix V has a left null-space of dimension, in general, larger that n. In other words, we have several options for the rows of $R(s)$. We must choose the rows in order to have a minimum degree, and $R(s)$ column reduced, fore more details see [39]. To facilitate this choice, it is convenient to solve (5.1) via the L echelon form. Column reducedness of factor $R(s)$ controls the introduction of MacMillan zeros at infinity. As pointed out in [11], this control is important for the spectral symmetric factor extraction which is the basis of our algorithm of J-spectral factorization. Nevertheless, notice that, even when $R(s)$ has no MacMillan zeros at infinity, if the factorization is inexact it has some zeros at infinity.

Example 5.2
Consider the matrix of Example 5.1. We want to extract a factor $R(s)$ containing the two finite zeros at $s = 0$. With the algorithms in Chapter 3 (Algorithm 3.3 for example) we obtain the two characteristics vectors associated to $s = 0$

$$v_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad v_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Then we construct the matrix

$$V = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

such that a basis of its left null-space allows to find factor $R(s)$:

$$\begin{bmatrix} R_0 & R_1 & R_2 \end{bmatrix} V = 0, \qquad \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} V = 0.$$

Notice that no pair of rows in the basis of the left null-space of V gives a factor

$$R(s) = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix}.$$

Choosing rows 1 and 4 we obtain a column reduced factor $R(s)$ which is the same as in Example 5.1. ∎

5.1.2 Extraction of the null-space

Now we extend the algorithm presented above to extract an $r \times n$ factor $R(s)$ sharing the same null-space as a square singular $n \times n$ polynomial matrix $A(s)$ of rank r and degree d. Suppose that a minimal basis of the null-space of $A(s)$ contains the vectors $v_i(s) = v_{i0} + v_{i1}s + \cdots + v_{i\delta_i}s^{\delta_i}$ for $i = 1{:}n-r$ such that $A(s)v_i(s) = 0$. Define

$$W_i = \begin{bmatrix} v_{i0} & \cdots & v_{i\delta_i} & & 0 \\ & \ddots & & \ddots & \\ 0 & & v_{i0} & \cdots & v_{i\delta_i} \end{bmatrix},$$

and build up the matrix $W = [W_1 \cdots W_{n-r}]$. We can show that a factor $R(s) = R_0 + R_1 s + \cdots + R_d s^d$ sharing the same null-space as $A(s)$ can be obtained by solving the system

$$\begin{bmatrix} R_0 & R_1 & \cdots & R_d \end{bmatrix} W = 0. \tag{5.2}$$

Now, rows of the left null-space of W are chosen such that factor $R(s)$ has minimal degree. One more time, the solution of (5.2) via the L echelon form facilitates this task.

Example 5.3
Consider the matrix

$$A(s) = \begin{bmatrix} s & 0 & 1 \\ s^2 & 0 & s \\ 2s - s^2 & 0 & 2 - s \end{bmatrix}.$$

We want to extract a factor $R(s)$ sharing the same right null-space as $A(s)$. With the algorithms in Chapter 4 (Algorithm 4.1 for example) we obtain the vectors

$$v_1(s) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad v_2(s) = \begin{bmatrix} -0.71 \\ 0 \\ 0.71s \end{bmatrix}$$

generating a minimal basis of the right null-space of $A(s)$. Then we construct matrix

$$W = [W_1 \quad W_2] = \left[\begin{array}{cc|ccc} 0 & 0 & -0.71 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.71 & 0 \\ 0 & 0 & 0 & -0.71 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.71 \end{array}\right]$$

such that a basis of its left null-space allows to find factor $R(s)$

$$\left[\begin{array}{cc} R_0 & R_1 \end{array}\right] W = 0, \qquad \left[\begin{array}{cccccc} 0 & 0 & 1 & 1 & 0 & 0 \end{array}\right] W = 0.$$

So, the minimum degree factor $R(s)$ is

$$R(s) = \left[\begin{array}{ccc} s & 0 & 1 \end{array}\right].$$

Notice that $R(s)$ has no zeros at infinity, so the factorization is exact. ∎

5.2 J-spectral factorization

The *J-spectral factorization* (JSF) problem for polynomial matrices is defined as follows. Let

$$A(s) = A_0 + A_1 s + \cdots + A_d s^d \tag{5.3}$$

be an $n \times n$ polynomial *para-Hermitian matrix* with real coefficients and degree d in the complex indeterminate s. Matrix $A(s)$ is para-Hermitian if $A(s) = A^T(-s)$ where T denotes the transpose. We want to obtain the factorization

$$A(s) = P^T(-s) J P(s) \tag{5.4}$$

where $P(s)$ is an $n \times n$ non-singular matrix whose spectrum[2] lie within the left half-plane, and where

$$J = \mathrm{diag}\{I_{n_+}, I_{n_-}, 0_{n_0}\} \tag{5.5}$$

is a signature matrix with $+1$, -1 and 0 along the diagonal, and such that $n_+ + n_- + n_0 = n$.

The J-spectral factorization of para-Hermitian polynomial matrices has important applications in control and systems theory, as described first in [114]. See e.g. [60] and [34] for comprehensive descriptions of applications in multivariable Wiener filtering, LQG control and H_∞ optimization.

In its most general form, the JSF problem applies to singular matrices. In this paragraph, we show that the problem of factorizing a singular matrix can be converted into the problem of factorizing a non-singular matrix. Alternatively, we can also seek a non-square spectral factor. If $A(s)$ has rank $r < n$, then $n_+ + n_- = r$ in signature

2. The set of all the zeros (eigenvalues) of a polynomial matrix [31], cf. Chapter 3 in this thesis.

matrix (5.5). The null-space of $A(s)$ is symmetric, namely if the columns of $Z(s)$ form a basis of the right null-space of $A(s)$, i.e. $A(s)Z(s) = 0$, then $Z^T(-s)A(s) = 0$. As pointed out in [92], the factorization of $A(s)$ could start with the extraction of its null-space. There always exists an unimodular matrix

$$U^{-1}(s) = [B(s) \quad Z(s)]$$

such that

$$U^{-T}(-s)A(s)U^{-1}(s) = \text{diag}\{\bar{A}(s), 0\} \tag{5.6}$$

where the $r \times r$ matrix $\bar{A}(s)$ is non-singular. Now, if $\bar{A}(s)$ is factorized as $\bar{A}(s) = \bar{P}^T(-s)\bar{J}\bar{P}(s)$, then the JSF of $A(s)$ is given by:

$$J = \text{diag}\{\bar{J}, 0\}, \quad P(s) = \text{diag}\{\bar{P}(s), I_{n-r}\}U(s).$$

Equivalently, if we accept $P(s)$ to be non-square, then we can eliminate the zero columns and rows in factorization (5.4) and obtain

$$A(s) = P^T(-s)JP(s), \quad J = \text{diag}\{I_{n_+}, -I_{n_-}\} \tag{5.7}$$

where $P(s)$ has size $r \times n$. The different properties of factorizations (5.4) and (5.7) are discussed later. What is important to notice here is that any singular factorization can be reduced to a non-singular one and then, the theory of non-singular factorizations can be naturally extended to the singular case.

5.2.1 Existence conditions

Suppose that non-singular para-Hermitian polynomial matrix $A(s)$ admits a factorization $A(s) = P^T(-s)JP(s)$ where constant matrix J has size $m \times m$. Let $\sigma[A(s)]$ be the spectrum of $A(s)$, then

$$m \geq m_0 = \max_{z \in i\mathbb{R}/\sigma[A(s)]} \mathcal{V}_+[A(z)] + \max_{z \in i\mathbb{R}/\sigma[A(s)]} \mathcal{V}_-[A(z)]$$

where $\mathcal{V}_+$ and $\mathcal{V}_-$ are, respectively, the number of positive and negative eigenvalues of a Hermitian matrix [86]. If $A(s)$ has no *constant signature* on the imaginary axis, i.e. the difference $\mathcal{V}_+[A(z)] - \mathcal{V}_-[A(z)]$ is not constant for all $z \in i\mathbb{R}/\sigma[A(z)]$, then $n < m_0 \leq 2n$, see the proof of Theorem 3.1 in [86]. On the other hand, if $A(s)$ has constant signature, then $m = m_0 = n$, and matrix J can be chosen as the unique square matrix given in (5.5) where $n_0 = 0$. It is shown in [86] that any para-Hermitian matrix $A(s)$ with constant signature admits a JSF. So, constant signature of $A(s)$ is the basic existence condition for JSF that we will assume in this thesis.

In [87] the authors give necessary and sufficient conditions for a self-adjoint polynomial matrix to have constant signature, see also [30]. These results can be extended to para-Hermitian polynomial matrices but this is out of the scope of this thesis. Some other works giving necessary and sufficient conditions for the existence of the JSF are [69] and [88]. There, the conditions are related to the existence of an stabilizing

solution of an associated Riccati equation. A deeper discussion of all the related results on the literature would be very extensive and also out of our objectives.

It is easy to see that if $A(s)$ is para-Hermitian, then the degrees δ_i for $i = 1{:}n$ of the n diagonal entries of $A(s)$ are even numbers. We define the diagonal leading matrix of $A(s)$ as

$$\mathcal{A} = \lim_{|s|\to\infty} = D^{-T}(-s)A(s)D^{-1}(s) \tag{5.8}$$

where $D(s) = \text{diag}\{s^{\frac{\delta_1}{2}}, \ldots, s^{\frac{\delta_n}{2}}\}$. We say that $A(s)$ is *diagonally reduced* if $\mathcal{A}$ exists and is non-singular. The JSF (5.4, 5.5) can be defined for both diagonally or non-diagonally reduced matrices. From [60] we say that the JSF of a diagonally reduced matrix $A(s)$ is *canonical* if $P(s)$ is column reduced with column degrees equal to half the diagonal degrees of $A(s)$, see also [29].

5.2.2 Algorithm

Once the finite zeros of $A(s)$ are given (cf. Section 3.1 in this thesis), we divide the problem of JSF into two major parts: first the computation of the eigenstructure of $A(s)$ achieved by the algorithms described in Chapters 3 and 4 (in particular Algorithms 3.3 and 4.1), and second the extraction of factor $P(s)$ using the algorithms of factor extraction described in Sections 5.1.1 and 5.1.2.

Algorithm 5.1 (`fact_spect`) *For a matrix $A(s)$ as in* (5.3) *with constant signature, this algorithm computes the JSF $A(s) = P^T(-s)JP(s)$. We consider that the set z of finite zeros of $A(s)$ is given.*

1. *If $r < n$, extract an $r \times n$ polynomial factor $R_n(s)$ sharing the same right null-space as $A(s)$. Solve the polynomial equation*

$$A(s) = R_n^T(-s)X_1(s)R_n(s)$$

where $X_1(s)$ is a full-rank $r \times r$ polynomial matrix containing only the finite zeros of $A(s)$ and some zeros at infinity. If $r = n$ let $X_1(s) = A(s)$.

2. *Extract a polynomial factor $R_f(s)$ containing the finite left half plane zeros of $A(s)$ and half of its finite zeros on the imaginary axis. Solve the polynomial equation*

$$X_1(s) = R_f^T(-s)X_2(s)R_f(s)$$

where $X_2(s)$ is a full-rank unimodular matrix.

3. *Extract from $X_2(s)$ a factor $R_\infty(s)$ containing half of its zeros at infinity. Solve the polynomial equation*

$$X_2(s) = R_\infty^T(-s)CR_\infty(s)$$

where C is a constant symmetric matrix.

4. *Finally with a Schur decomposition we factorize C as $C = U^T JU$. At the end we obtain the expected spectral factor*

$$P(s) = UR_\infty(s)R_f(s)R_n(s).$$

Reliable algorithms to solve polynomial equations of the type $A(s)X(s)B(s) = C(s)$ are based on iteratively solving linear systems of equations, see the documentations of functions `axb`, `xab` and `axbc` of the Polynomial Toolbox for Matlab [84]. Factorization of matrix C in step 4 is ensured by the Schur algorithm [33].

When $A(s)$ is positive definite, factorization in step 3 is always exact and factor $P(s)$ has always half of the zeros at infinity of $A(s)$, see [11]. If $A(s)$ is indefinite, there are cases where an exact factorization in step 3 is not possible. For instance consider that $X_2(s)$ results in the unimodular matrix of Example 3.3 in [60]. In that cases notice that $X_2(s)$ can have a factorization with a non-minimal degree spectral factor[3]. When $A(s)$ is diagonally reduced, i.e. without MacMillan zeros at infinity [11], this non-minimality in the factorization of $X_2(s)$ implies that $P(s)$ is not canonical in the sense of [60]. Nevertheless notice that, even if we cannot conclude about the diagonal reducedness of $A(s)$, the non-minimality in the factorization of $X_2(s)$ can be detected. So, in general we can say that a JSF is not canonical if $P(s)$ has more than half of the zeros at infinity of $A(s)$.

Example 5.4
Consider the matrix

$$A(s) = \begin{bmatrix} 0 & 12 - 10s - 4s^2 + 2s^3 \\ 12 + 10s - 4s^2 - 2s^3 & -16s^2 + 4s^4 \end{bmatrix}$$

which has finite zeros $\{-1, 1, -2, 2, -3, 3\}$ and two zeros at infinity. First we extract the factor

$$R_f(s) = \begin{bmatrix} 3 + 4s + s^2 & 0 \\ 0 & 2 + s \end{bmatrix}$$

containing the negative finite zeros of $A(s)$, and we build the factorization

$$A(s) = R_f^T(-s)X_2(s)R_f(s) = R_f^T(-s) \begin{bmatrix} 0 & 2 \\ 2 & -4s^2 \end{bmatrix} R_f(s).$$

Notice that $X_2(s)$ is unimodular and has 4 zeros at infinity. The only column reduced factor containing 2 zeros at infinity that we can extract from $X_2(s)$ is given by $R_\infty(s) = \text{diag}\{1, s^2\}$. Notice that $R_\infty(s)$ has also two finite zeros at 0, so the factorization is inexact. With Algorithm 3.2 of [60][4] we can factorize $X_2(s)$ as $X_2(s) = R_\infty^T(-s)\text{diag}\{1, -1\}R_\infty(s)$ with

$$R_\infty(s) = \begin{bmatrix} 1 & 1 - s^2 \\ 1 & -1 - s^2 \end{bmatrix}.$$

Finally, the spectral factor $P(s)$ is given by

$$P(s) = \begin{bmatrix} 3 + 4s + s^2 & 2 + s - 2s^2 - s^3 \\ 3 + 4s + s^2 & -2 - s - 2s^2 - s^3 \end{bmatrix}.$$

3. In the sense that the degree of $X_2(s)$ is not the double of the degree of $R_\infty(s)$.
4. Numerical reliability of this algorithm is not guaranteed.

Notice that $\mathcal{A}$ given by (5.8) does not exist, we cannot conclude about the diagonally reducedness of $A(s)$ but we can say that the factorization is not canonical according to our definition. Spectral factor $P(s)$ has finite zeros $\{-1,-2,-3\}$ but also 3 zeros at infinity, namely, more than half of the zeros at infinity of $A(s)$. ∎

When $A(s)$ is rank deficient, we extract a non-square factor $R_n(s)$ in step 1 of Algorithm 5.1. Equivalently we can extract a square factor as in (5.6) with the reliable methods presented in [38]. Differences between both approaches are showed in the following example.

Example 5.5
Consider the singular matrix

$$A(s) = \begin{bmatrix} s^2+s^8 & s+s^7 & s^4 \\ -s-s^7 & -1-s^6 & -s^3 \\ s^4 & s^3 & 1 \end{bmatrix}.$$

Matrix $A(s)$ has rank 2 and 14 zeros at infinity. A right null-space basis is given by $v(s) = [-0.71\ 0.71s\ 0]^T$. A factor $R_n(s)$ sharing the same right null-space as $A(s)$ is given by

$$R_n(s) = \begin{bmatrix} 0 & 0 & 1 \\ s & 1 & 0 \end{bmatrix},$$

so we have the factorization $R_n^T(-s)X_2(s)R_n(s) = A(s)$ with

$$X_2(s) = \begin{bmatrix} 1 & s^3 \\ -s^3 & -1-s^6 \end{bmatrix}.$$

Matrix $X_2(s)$ has only zeros at infinity and it can be exactly factorized as $X_2(s) = R_\infty^T(-s)\text{diag}\{1,-1\}R_\infty(s)$ with

$$R_\infty(s) = \begin{bmatrix} 1 & s^3 \\ 0 & 1 \end{bmatrix}.$$

Therefore the spectral factor $P(s)$ is given by

$$P(s) = R_\infty(s)R_n(s) = \begin{bmatrix} s^4 & s^3 & 1 \\ s & 1 & 0 \end{bmatrix}.$$

Notice that $\mathcal{A}$ given by (5.8) has rank 1, we cannot conclude about the diagonal reducedness of $A(s)$ but we can say that the factorization is canonical according to our definition. Spectral factor $P(s)$ has 7 zeros at infinity, namely, half of the zeros at infinity of $A(s)$. On the other hand, if at step 1 of Algorithm 5.1 we build a square factor $R_n(s)$ such that $R_n^T(-s)X_2(s)R_n(s) = A(s)$ with

$$X_2(s) = \begin{bmatrix} -1 & s^3 & 0 \\ -s^3 & 1+s^6 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$R_n(s) = \begin{bmatrix} s+s^7 & 1+s^6 & s^3 \\ s^4 & s^3 & 1 \\ 1.4+0.71s^6 & 0.71s^5 & 0.71s^2 \end{bmatrix},$$

then factor $P(s)$ is given by

$$P(s) = \begin{bmatrix} s^4 & s^3 & 1 \\ s & 1 & 0 \\ 1.4 + 0.71s^6 & 0.71s^5 & 0.71s^2 \end{bmatrix}.$$

In this case notice that the factorization is not canonical according to our definition. Factor $P(s)$ now has full rank, but this implies the introduction of zeros at infinity. $P(s)$ has 18 zeros at infinity, namely, more than half of the zeros at infinity of $A(s)$. ■

5.3 Algorithmic analysis

Here we are interested in underlining the contributions of our JSF algorithm with respect to other useful algorithms in the literature. As pointed out in Section 2.3 of this thesis, these JSF algorithms can be classified as state-space algorithms or polynomial algorithms. Unlike state-space algorithms for the eigenstructure, state-space algorithms for the JSF are not based on the linearization of matrix $A(s)$. The state-space methods usually relate the problem of JSF to the stabilizing solution of an algebraic Riccati equation. One of the first contributions was [103], where an algorithm for the standard sign-definite case ($J = I$) is presented. The evolution of this kind of methods is surveyed in [95] and references inside. This paper, based on the results of [102], describes and algorithm that can deal with indefinite para-Hermitian matrices (with constant signature) and with zeros along the imaginary axis. The case of singular matrices is tackled in [96] only for the discrete case. The popularity of the state-space methods is related with its good numerical properties, cf. Chapter 2 of this thesis. There exist several numerical reliable algorithms to solve the Riccati equation, see for example [1, 9]. On the negative side, sometimes the reformulation of the polynomial problem in state-space requires elaborated preliminary steps [60] and some concepts and particularities of the polynomial problem are difficult to recover from the new representation.

On the other hand, the advantage of polynomial methods is their conceptual simplicity and straightforward application to the polynomial matrix, resulting, in general, in faster algorithms (cf. Sections 3.3.1 and 4.2.1). On the negative side, the polynomial methods are often related with elementary operations over the ring of polynomials, and it is well known that these operations are numerically unstable (cf. Section 2.2.1 in this thesis). Our JSF algorithm follows the idea of symmetric factor extraction used first by Davis for the standard spectral factorization [18]. Davis' algorithm was improved in [11] and [60] under the assumption that the polynomial matrix is not singular and has no zeros on the imaginary axis. Moreover, the computations are still based on unreliable elementary polynomial operations. Therefore, our contributions with respect to these algorithms are twofold:

1. Generality: Our algorithm can handle the most general case of a possibly singular, indefinite para-Hermitian matrix, with no assumptions on its diagonally reducedness or the locations of its zeros along the imaginary axis. As far as we know, the only polynomial method which can deal with this general case is the diagonalization algorithm of [60]. This algorithm is based on iterative elementary polynomial operations and thus, it can be quite sensitive to numerical round-off errors, cf. Example 2.5.
2. Stability: No elementary polynomial operations are needed. Our algorithm is based on numerically reliable operations only, namely computation of the null-spaces of constant block Toeplitz matrices along the lines sketched in Chapters 3 and 4, factor extraction as described in Sections 5.1.1 and 5.1.2, see also [39], as well as solving linear polynomial matrix equations[5].

Direct comparisons between our algorithm and the algorithm in [95] are difficult to achieve. We claim that both algorithms improve classical methods in their own style. On the one hand, the algorithm in [95] avoids some preparatory steps usually necessary in the state-space approach via a minimal state-space dimension. So, it brings improvements in terms of computational effort. On the other hand, our algorithm avoids elementary operations over polynomials which are the basis of standard polynomial methods. So, it brings improvements in terms of numerical stability.

Even if there exist satisfactory results on computing the finite zeros of a polynomial matrix, cf. Section 3.1 in this thesis, in practice, a real advantage of the state-space methods over our algorithm is that they do not need to compute these finite zeros beforehand. Nevertheless notice that, by exploiting the information on the eigenstructure of the polynomial matrix, further insight is lend into several cases of JSF. Consider for example the non-canonical factorization: we have seen that it is related with an inexact extraction of the infinite structure of the polynomial matrix. Another advantage of using the polynomial eigenstructure is that our algorithm can be applied almost verbatim to the discrete time case[6]. In fact, a discrete time polynomial matrix can be considered as a continuous time rational matrix. The structure at infinity of the discrete polynomial matrix is mapped into the infinite structure and the structure at $s = 0$ of the associated rational matrix. This fact explains the presence of the spurious zeros [96] but also allows us to handle them naturally. For the discrete time case, factorization in step 2 of Algorithm 5.1 presented here can be achieved in two different ways: (1) by extracting a factor containing the finite zeros at $s = 0$ (also zeros at infinity of the discrete time polynomial matrix) and (2) by extracting a factor containing only zeros at infinity. In the case (1) we recover the result of Algorithm 1 in [96] with the presence of spurious zeros, and in the case (2) we recover the result of Algorithm 2 in [96].

5. Nevertheless, notice that global backward stability of Algorithm 5.1 cannot be guaranteed. Consider for example the null-space factor extraction step: the null-space computed by Algorithm 4.1 is used as input of the factor extraction algorithm described in section 5.1.2. So, it is the sensibility of the null-space structure problem that determines how the backward error of factor extraction algorithm is projected into the coefficients of the analyzed polynomial matrix, and as we saw in Chapter 4, computing the null-space structure is an ill-posed problem.

6. As pointed out in [96] this is not possible with the state-space methods.

Chapitre 6

Conclusions

6.1 Contexte scientifique

La puissance des ordinateurs modernes a permis à la science et à l'ingénierie d'avancer énormément ces dernières décades. En commande, la simulation des problèmes et conception des solutions à l'aide de l'ordinateur sont devenues des pratiques communes. Pratiques qui, naturellement, sont possibles grâce au travail de développement de logiciels.

Ce développement de logiciels comprend plusieurs aspects dont la plupart sont transparents aux utilisateurs. Dans cette thèse nous avons étudié des aspects numériques: la limitation en la précision avec laquelle les nombres réels sont représentés par l'ordinateur pose des problèmes numériques qui doivent être traités soigneusement pour obtenir des résultats adéquats.

L'analyse numérique est devenue la discipline qui permet d'étudier ces problèmes numériques et les algorithmes pour les résoudre. Ainsi les sciences et disciplines mathématiques ont développé leurs branches d'analyse numérique dont l'algèbre linéaire numérique est une des plus riches.

Ce développement de l'algèbre linéaire numérique a permis de résoudre beaucoup de problèmes en commande, formulés via l'approche espace d'état. Actuellement l'abondance de logiciels de commande assistée par ordinateurs (CACSD) a rendu l'approche espace d'état la méthode de prédilection quand des résultats numériques sont requis.

D'autres approches à la commande comme l'approche polynomiale, très développée au niveau théorique, sont plus limités numériquement. Certains logiciels de CACSD correspondants sont basés sur le calcul symbolique qui implique un temps d'exécution et une demande en mémoire plus importants. Mais surtout, quand des coefficients inexacts sont traités, les résultats obtenus par le calcul symbolique ne sont pas utilisables en pratique (ils peuvent être donnés comme des fractions de nombres avec des centaines de chiffres par exemple).

Quand un problème sur des polynômes ou des matrices polynomiales est numériquement réductible, c'est-à-dire qu'il peut être reformulé comme un problème équivalent sur des matrices et vecteurs constants, l'utilisation des méthodes numériques résulte en des algorithmes plus performants. Dans cette thèse nous avons montré que les opérations élémentaires sur les polynômes, cause de l'instabilité numérique des méthodes polynomiales classiques, peuvent être évitées grâce à cette reformulation du problème original.

L'instabilité des opérations élémentaires est donc remplacée par la stabilité des méthodes numériques de l'algèbre linéaire. La solution du problème constant équivalent est alors traduite en termes du problème polynômial original mais malheureusement, l'exactitude de cette solution n'est pas assurée. Deux aspects affectent cette qualité:

1. la manière de laquelle l'erreur arrière sur le problème constant équivalent est reflétée sur les coefficients du problème polynômial original,
2. la manière de laquelle la solution du problème original répond à cette perturbation sur les données d'entrée.

Les deux aspects ci-dessus impliquent deux analyses basiques: l'analyse de l'erreur et l'analyse de la sensibilité. Ces analyses sont bien comprises dans le cas de l'algèbre linéaire numérique, mais ils ne peuvent pas être appliqués de la même façon aux problèmes non linéaires comme les polynômes. La consolidation d'une algèbre non linéaire numérique reste donc une grande lacune à combler pour que les problèmes numériques sur les polynômes et les matrices polynomiales soient résolus complètement. Des travaux très récents sur l'algèbre polynomiale numérique semblent indiquer que certains problèmes polynômiaux mal posés peuvent être résolus de manière satisfaisante.

Ainsi, notre point de vue est qu'il est une question de temps pour qu'il existe davantage de logiciels numériques de qualité pour les polynômes et les matrices polynomiales, et pour que le développement des logiciels de CACSD pour l'approche polynomiale soit comparable à celui de l'approche espace d'état.

6.2 Contribution et perspectives

Cette thèse a été notre contribution à l'objectif décrit dans le paragraphe précédent. Notre motivation ont été les problèmes en commande formulés dans l'approche polynomiale. Nous avons développé des algorithmes numériques pour l'obtention de la structure propre d'une matrice polynomiale et pour sa factorisation J-spectrale. Globalement, l'analyse des algorithmes a été basée sur les propriétés des méthodes de l'algèbre linéaire numérique. L'analyse du point de vue polynômial est un travail qui dépasse les portées de cette thèse. Cependant nous avons montré qu'une analyse rigoureuse peut être effectuée également pour les algorithmes sur les matrices polynomiales et nous avons établi clairement plusieurs lignes le long desquelles les résultats de cette thèse peuvent être étendus.

Ci-dessous nous présentons un résumé des contributions techniques de cette thèse ainsi que des perspectives de travaux futurs:

Nous avons considéré comme hypothèse initiale de nos algorithmes que les zéros finis de la matrice polynomiale analysée sont donnés. Le problème de l'obtention des zéros d'un polynôme (à coefficients scalaires ou matricielles) est un problème pouvant bien mériter une thèse. Ici nous avons seulement décrit brièvement les techniques les plus récentes. Nous avons vu que les techniques appliquées aux polynômes scalaires sont plus avancées et qu'elles permettent de résoudre le problème même pour les cas des racines multiples, qui est un problème mal posé. L'extension de ces techniques aux matrices polynomiales est une ligne de travail futur.

L'hypothèse que les zéros finis sont donnés représente une limitation de notre approche surtout dans le cas de la factorisation J-spectrale de matrices polynomiales, étudiée dans le Chapitre 5. Nous avons vu dans ce chapitre qu'ils existe d'autres algorithmes qui peuvent s'affranchir de cette hypothèse. Cependant, nous avons vu aussi que le fait de considérer la structure des zéros de la matrice polynomiale permet une analyse plus fine et directe des différents cas de la factorisation (comme les factorisations non canoniques ou les factorisations à temps discret), ainsi qu'une application directe de notre algorithme sans aucune modification particulier pour chaque cas. Voir [125] pour une publication issue de cette thèse, décrivant l'algorithme développé pour la factorisation J-spectrale.

Dans les Chapitres 3 et 4 nous avons développé des algorithmes Toeplitz pour obtenir les structures finie et infinie d'une matrice polynomiale ainsi que son espace nul. Ces algorithmes sont aussi décrits dans [124], publication issue de cette thèse. Nous avons montré qu'une expression exacte du nombre d'opérations réalisées par l'algorithme est difficile à obtenir car de nombreux paramètres interviennent. Voir aussi [123], où ces résultats ont été publiés.

En faisant cette analyse nous avons aussi montré que nos algorithmes peuvent être plus efficaces, au niveau complexité algorithmique, que les algorithmes classiques de l'approche par faisceaux. Un avantage important des algorithmes Toeplitz est la faible dépendance du nombre d'opérations exécutées par rapport au degré de la matrice analysée.

Une autre caractéristique qui favorise la performance des algorithmes Toeplitz est la formulation par blocs. Avec cette formulation nous pouvons profiter du calcul effectué lors d'un pas pour accélérer le calcul effectué au pas suivant. Nous avons montré que cette croissance est freinée, malgré le fait qu'à chaque pas la matrice Toeplitz grandit.

Un des grands avantages de l'approche Toeplitz et des algorithmes développés dans cette thèse est qu'ils obtiennent, avec le même effort de calcul, non seulement les indices structurels mais aussi les vecteurs propres associés aux zéros de la matrice polynomiale ou les vecteurs qui forment son espace nul. Cet avantage est clairement illustré dans les exemples du Chapitre 4. Sont mis alors en évidence les grands temps de calcul requis par les algorithmes de faisceaux pour calculer les vecteurs dans la base de l'espace nul de la matrice analysée.

Dans le Chapitre 4 nous avons aussi comparé les performances des factorisations orthogonales classiques et des factorisations dites rapides basées sur l'application de la méthode généralisée de Schur. Ces résultats ont été présentés dans [126], publication issue de cette thèse. L'idée de diminuer l'ordre de la complexité algorithmique à l'aide des méthodes rapides est attractive, malheureusement nous avons montré que cela n'est pas toujours possible. Pour que les méthodes rapides assurent une bonne performance, quelques conditions sont nécessaires. Ces conditions, dépendant de la structure propre de la matrice polynomiale analysée, ne sont pas toujours vérifiées.

Dans [128], article soumis et encore en révision, nous concentrons tous les résultats concernant le calcul de l'espace nul des matrices polynomiales.

Nous avons montré que l'approche Toeplitz permet aussi d'analyser facilement quelques propriétés de la structure propre des matrices polynomiales. Dans le Chapitre 3, nous avons déduit des conditions nécessaires et suffisantes pour que le degré de MacMillan à l'infini d'une matrice polynomiale soit nul, c'est-à-dire des conditions d'absence de zéros de MacMillan à l'infini d'une matrice polynomiale. Nos conditions sont purement algébriques et basées sur le calcul de rangs de matrices Toeplitz par blocs. Ces conditions peuvent remplacer les conditions classiques et seulement suffisantes, à savoir: une matrice polynomiale n'a pas des zéros de MacMillan à l'infini si elle est réduite par colonnes, ou réduite par lignes, ou diagonalement réduite. Voir aussi [40], publication issue de cette thèse.

Dans le Chapitre 5, en définissant les zéros à l'infini comme les zéros en $s = 0$ de la matrice polynomiale duale, nous avons étudié le problème de l'extraction d'un facteur contenant une partie de la structure propre de la matrice polynomiale analysée. Nous avons déduit des conditions simples pour qu'une telle factorisation soit exacte. Nous avons montré que dans les cas de factorisations inexactes, des zéros à l'infini ou des zéros à l'origine sont introduits. En se basant sur ces résultats, nous étendons l'idée de factorisation spectrale non canonique pour le cas de matrices non nécessairement diagonalement réduites.

Finalement, dans le Chapitre 4 nous avons montré que le rang des matrices polynomiales peut être obtenu comme un sous-produit des algorithmes Toeplitz. Ainsi nous avons démontré que l'approche Toeplitz est aussi générale que l'approche par faisceaux, et qu'elle peut être considérée comme une alternative fiable à cette dernière pour déterminer toute la structure propre d'une matrice polynomiale.

Comme mentionné ci-dessus, l'analyse de stabilité des algorithmes développés s'est basée sur les propriétés numériques des méthodes utilisées dans la solution du problème constant équivalent, c'est-à-dire le calcul des espaces nuls des matrices Toeplitz par blocs. La stabilité de ces méthodes, et en particulier de la factorisation LQ (duale de QR), est garantie et bien étudiée dans la littérature de l'algèbre linéaire numérique. De même il est facile de démontrer (voir Chapitre 3) la stabilité de notre formulation par blocs en la considérant comme une succession particulière de transformations de Householder.

Une analyse simple permet de transformer l'erreur arrière sur les coefficients des matrices Toeplitz en l'erreur arrière sur les coefficients de la matrice polynomiale

analysée. Cette analyse consiste tout simplement à donner à la perturbation sur les coefficients des matrices Toeplitz une structure Toeplitz par blocs, voir Chapitre 4. De cette façon on démontre que l'erreur arrière de nos algorithmes de calcul de la structure propre est bornée. Ces résultats ont aussi été présentés dans [127].

Pour une structure fixée, i.e. un nombre donné de chaînes de vecteurs propres de longueurs données, ou une dimension donnée de l'espace nul avec degrés de ses vecteurs donnés, une bonne exactitude de la solution obtenue par nos algorithmes est aussi assurée par les propriétés numériques des méthodes utilisées. Déterminer la structure elle-même est par contre un problème mal posé, comme beaucoup de problèmes avec les polynômes.

Dans cette thèse nous avons mentionné quelques techniques de régularisation pour le problème des racines d'un polynôme. Ces techniques sont basées sur le raffinement itératif en considérant la géométrie du problème. L'extension de ces techniques à des géométries plus compliquées pour régulariser d'autres problèmes polynômiaux, comme notre problème de la structure propre, est une importante ligne de travail futur. Similairement, nous espérons qu'avec l'étude de la géométrie du problème, nous puissions aussi déterminer des expressions plus exactes de l'erreur arrière introduite dans les coefficients de la matrice analysée. Nous espérons également pouvoir développer des techniques de scaling pour améliorer le conditionnement d'une matrice polynomiale donnée, pour le problème du calcul de sa structure propre.

Bibliographie

[1] Abou-Kandil H., Freiling G., Jank G. and Ionescu V., *Matrix Riccati Equations in Control and Systems Theory*. Birkhäuser Verlag, Berlin, 2003.

[2] Ackermann J., *Der Entwurf Linearer Regelungssysteme im Zustandraum*. Regelungstechnik und Prozess-Datenverarbeitung, 7:297–300, 1972.

[3] Anderson E., Bai Z., Bischof C., Blackford S., Demmel J., Dongarra J., Du Croz J., Greenbaum A., Hammarling S., McKenney A. and Sorensen D., *LAPACK Users' Guide*. SIAM, Philadelphia, 1999.

[4] Antsaklis P. J. and Gao Z., *Polynomial and rational matrix interpolation: theory and control applications*. Int. J. Control, 58:349–404, 1993.

[5] Basilio J. C. and Moreira M. V., *A robust solution of the generalized polynomial Bezout identity*. Linear Algebra Appl., 385:287–303, 2004.

[6] Beelen Th. G. J. and Veltkamp G. W., *Numerical computation of a coprime factorization of a transfer function matrix*. Systems Control Lett., 9:281–288, 1987.

[7] Bhattacharyya S. P., *Frequency domain conditions for disturbance rejection*. IEEE Trans. Automat. Control, 25(6):1211–1213, 1980.

[8] Bini D. and Pan V., *Polynomial and Matrix Computations, Vol 1: Fundamental Algorithms*. Birkhauser, Boston, 1994.

[9] Bittanti S., Laub A. J. and Willems J. C. (Eds.), *The Riccati Equation*. Springer-Verlag, Berlin, 1991.

[10] Boettcher P., *Frequently asked questions in newsgroup* `comp.soft-sys.matlab`. MIT, Massachussetts, 2000.

[11] Callier F. M., *On polynomial matrix spectral factorization by symmetric extraction.* IEEE Trans. Autom. Control, 30(5):453-464, 1985.

[12] Callier F. M. and Desoer C. A., *Multivariable Feedback Systems*. Springer-Verlag, New York, 1982.

[13] Chabert J. L. (Ed.), *A History of Algorithms – From the Pebble to the Microchip*. Springer-Verlag, Milan, 1999.

[14] Chaitin-Chatelin F. and Frayssé V., *Lectures on finite precision computations*. SIAM, Philadelphia, 1996.

[15] Chan T. F., *Rank revealing QR Factorizations*. Linear Algebra Appl., 88/89:67–82, 1987.

[16] Cipra B. A., *The best of the 20th century: Editors name top 10 algorithms*. SIAM News, 33(4):1–2, 2000.

[17] Datta B. N. (Ed.), *Linear Algebra and its Role in Linear Systems Theory*. AMS Contemporary Mathematics Series, Providence, 1985.

[18] Davis M. C., *Factoring the spectral matrix*. IEEE Trans. Automat. Contr., 8:296–305, 1963.

[19] Dedieu J. P. and Tisseur F., *Perturbation theory for homogeneous polynomial eigenvalue problems*. Linear Algebra Appl., 358:71–94, 2003.

[20] Demmel J. and Dongarra J., *ST-HEC: reliable and scalable software for linear algebra computations on high-end computers*. LAPACK Working Note 164, University of California at Berkeley and University of Tennessee at Knoxville, Feb. 2005.

[21] Descusse J. and Dion J. M., *On the structure at infinity of linear square decoupled systems*. IEEE Trans. Automat. Control, 27(4):971–974, 1982.

[22] Dongarra J. and Sullivan F., *Guest editors introduction to the top 10 algorithms*. Computing in Science and Engineering, 2(1):22–23, 2000.

[23] A. Edelman and H. Murakami, *Polynomial roots from companion matrix eigenvalues*. Math. Comput., 64:763–776, 1995.

[24] Fan H. Y., Lin W. W. and Van Dooren P., *A note on optimal scaling of second order polynomial matrices*. SIAM J. Matrix Anal. Appl., 26:252–256, 2004.

[25] Forney G. D., *Minimal bases of rational vector spaces with applications to multivariable linear systems*. SIAM J. Control Optim., 13:493–520, 1975.

[26] Frisk E., *Residual Generation for Fault Diagnosis*. Ph.D. Thesis manuscript 716, Linköping University, Sweden, 2001.

[27] Gantmacher F. R., *Theory of Matrices*. Chelsea, New York, 1959.

[28] Gleick J., *A bug and a crash. Sometimes a bug is more than a nuisance*. New York Times Magazine, 1 December 1996.

[29] Gohberg I. and Kaashoek M.A., *Constructive methods of Wiener-Hopf factorization*. Birkhäuser, Basel, 1986.

[30] Gohberg I., Lancaster P. and Rodman L., *Factorization of self-adjoint matrix polynomials with constant signature*. Linear and Multilinear Algebra, 11:209–224, 1982.

[31] Gohberg I., Lancaster P. and Rodman L., *Matrix Polynomials*. Academic Press, New York, 1982.

[32] Goldberg D., *What every computer scientist should know about floating-point arithmetic*. ACM Computing Surveys, 23(1):5–48, 1991.

[33] Golub G. H. and Van Loan C. F., *Matrix Computations*. Johns Hopkins University Press, New York, 1996.

[34] Grimble M. and Kučera V. (Eds.), *A polynomial approach to H_2 and H_∞ robust control design*. Springer-Verlag, London, 1996.

[35] Hadamard J., *Sur les problèmes aux dérivées partielles et leur signification physique*. Princeton University Bulletin, 49–52, 1902.

[36] Hämmerlin G. and Hoffmann K. H., *Numerical Mathematics*. Springer-Verlag, New York, 1999.

[37] Henrion D., *Reliable Algorithms for Polynomial Matrices*. Ph.D. Thesis, Institute of Information Theory and Automation, Academy of Sciences of the Czech Republic, Prague, 1998.

[38] Henrion D. and Šebek M., *Reliable numerical methods for polynomial matrix triangularization*. IEEE Trans. Automat. Control, 44:497–508, 1999.

[39] Henrion D. and Šebek M., *An algorithm for polynomial matrix factor extraction*. Int. J. Control, 73(8):686–695, 2000.

[40] Henrion D. and Zúñiga J. C., *Detecting infinite zeros in polynomial matrices*. To appear in IEEE Trans. Circuits Systems, 2005.

[41] Herget C. J. and Laub A. J. (Ed.), *Special Issue on Computer-Aided Design of Control Systems*, IEEE Control Systems Magazine, 2(4), 1982.

[42] Higham N. J., *Analysis of the Cholesky decomposition of a semi-definite matrix*. Reliable Numerical Computations, Oxford University Press, 161–185, 1990.

[43] Higham N. J., *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, 1996.

[44] Higham D. J and Higham N. J., *Structured backward error and condition of generalized eigenvalue problems*. SIAM Matrix Anal. Appl., 20(2):493–512, 1998.

[45] Higham N. J., Konstantinov M. and Mehrmann V., *Sensitivity of computational control problems*. IEEE Control Systems Magazine, 24:28–43, 2004.

[46] Higham N. J., Mackey D. S. and Tisseur F., *The conditioning of linearizations of matrix polynomials*. Numerical Analysis report no. 456, Manchester Centre for Computational Mathematics, UK, April 2005.

[47] Hromčík M., *Numerical algorithms for polynomial factorizations*. Ph.D. Thesis, Department of Control Engineering, Czech Technical University in Prague, Czech Rep., 2004.

[48] Hunt K. J. (Ed.), *Polynomial Methods in Optimal Control and Filtering*. IEE Control Engineering Series 49, Peter Peregrinus, London, 1993.

[49] Ipsen I. C. F., *Accurate eigenvalues for fast trains*. SIAM News, 37(9), Nov. 2004.

[50] Kailath T., *Linear Systems*. Prentice Hall, Englewood Cliffs, 1980.

[51] Kailath T. and Sayed A. H., *Displacement Structure: theory and applications*. SIAM Review, 37:297-386, 1995.

[52] Kailath T. and Sayed A. H., *Fast Reliable Algorithms for Matrices with Structure*. SIAM, Philadelphia, 1999.

[53] Kautsky J., Nichols N. K. and Van Dooren P. M., *Robust pole assignment in linear state feedback*. Internat. J. Control, 41(5):1129–1155, 1985.

[54] Kraffer F., *Polynomial matrix to state space conversion without polynomial reduction*. IEEE Mediterranean Symposium on Control and Automation, Chania, Greece, 1996.

[55] Kraffer F., *State-space algorithms for polynomial matrix operations*. Amarilla no. 135, Department of Electrical Engineering of CINVESTAV, Mexico, D.F., 1998.

[56] Kressner D., *Numerical Methods for Structured Matrix Factorizations*. Ph.D. thesis, Faculty of Mathematics, Technical University of Chemnitz, Germany, 2001.

[57] Kressner D. and Van Dooren P. M., *Factorizations and linear system solvers for matrices with Toeplitz structure*. SLICOT Working Notes, 2000.

[58] Kučera V., *Discrete Linear Control: The Polynomial Equation Approach*. John Wiley and Sons, Chichester, 1979.

[59] Kučera V., *Diophantine equations in control–a survey*. Automatica, 29:1361–1375, 1993.

[60] Kwakernaak H. and Šebek M., *Polynomial J-spectral factorization*. IEEE Trans. Autom. Control, 39(2):315–328.

[61] Laub A. J., *Numerical linear algebra aspects of control design computations*. IEEE Trans. Automat. Control, 30:97–108, 1985.

[62] Lawson C. L., Hanson R. J., Kincaid D. R. and Krogh K. T., *Basic Linear Algebra Subprograms for FORTRAN usage*. ACM Trans. Math. Software, 5:308–323, 1979.

[63] Lemonnier D. and Van Dooren P., *Optimal scaling of companion pencils for the QZ-algorithm*. SIAM Conference on Applied Linear Algebra, Williamsburg, VA, 2003.

[64] Lemonnier D. and Van Dooren P., *Optimal scaling of block companion pencils*. International Symposium on Mathematical Theory of Networks and Systems, Leuven, Belgium, 2004.

[65] Loiseau J. J., *Sur la modification de la structure à l'infini par un retour d'état statique*. SIAM J. Control Optim., 26:251–273, 1988.

[66] Mackey D. S., Mackey N., Mehl C. and Mehrmann V., *Vector spaces of linearizations for matrix polynomials*. Numerical Analysis report no. 464, Manchester Centre for Computational Mathematics, UK, April 2005.

[67] Malabre M. and Kučera V., *Infinite structure and exact model matching problem: a geometric approach.* IEEE Trans. Automat. Control, 29(3):266–268, 1984.

[68] The MathWorks Inc., *Matlab*, Version 7 (Release 14), Natick, Pennsylvania, 2004.

[69] Meinsma G., *J-spectral factorization and equalizing vectors.* Systems Control Lett., 25:243–249, 1995.

[70] Miminis G. and Paige C. C., *An algorithm for pole assignment of time invariant linear systems.* Internat. J. Control, 35:341–345, 1982.

[71] Miminis G. and Paige C. C., *A direct algorithm for pole assignment of time invariant multi-input linear systems using state feedback.* Automatica, 24:343–356, 1988.

[72] Moler C. B. and Stewart G. W., *An algorithm for generalized matrix eigenvalue problems.* SIAM J. Numer. Anal., 10:241–256, 1973.

[73] Moler C. B., *Floating points. IEEE standard unifies arithmetic model.* Cleve's corner, Matlab newsletter News & Notes, Fall 1996.

[74] Moler C. B. and Van Loan C. F., *Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later.* SIAM Review, 45(1):801–837, 2003.

[75] Morse A. S., *Structural invariants of linear multivariable systems.* SIAM J. Control Optim., 11:446–465, 1973.

[76] Munro N. (Ed.), *Symbolic methods in control and system analysis and design.* IEE Control Enginnering Series 56, London, UK. 1999.

[77] Neven W. H. L. and Praagman C., *Column reduction of polynomial matrices.* Linear Algebra Appl., 188:569–589, 1993.

[78] Orozco J. L., Ruiz-León J., Begovich O. and Zúñiga J. C., *Transformation of a linear multivariable system to its semi-canonical Morse form.* Congreso Latinoamericano de Control Automático, City of the Habana, Cuba, 2004.

[79] Osborne E. E., *On pre-conditioning of matrices.* J. ACM, 7:338–345, 1960.

[80] Overton M. L., *Numerical computation with IEEE floating point arithmetic.* SIAM, Philadelphia, 2001.

[81] Parlett B. N. and Reinsch C., *Balancing a matrix for calculation of eigenvalues and eigenvectors.* Numer. Math., 13:293–304, 1969.

[82] Petkov P., Christov N. and Konstantinov M., *Computational Methods for Linear Control Systems.* Prentice Hall, Englewood Cliffs, 1991.

[83] Polderman J. W. and Willems J. C., *Introduction to Mathematical Systems Theory: a Behavioral Approach.* Springer-Verlag, 1998.

[84] PolyX Ltd., *The Polynomial Toolbox for Matlab*, Version 2.5, Prague, Czech Republic, 2000.

[85] Press W. H., Flannery B. P. and Teukolsky S. A., *Numerical Recipes in Fortran: The Art of Scientific Computing.* Cambridge University Press, 1992.

[86] Ran A. C. M. and Rodman L., *Factorization of matrix polynomials with symmetries.* SIAM J. Matrix Anal. Appl., 15(3):845–864, 1994.

[87] Ran A. C. M. and Zizler P., *On self-adjoint matrix polynomials with constant signature.* Linear Algebra Appl., 259:133–153, 1997.

[88] Ran A. C. M., *Necessary and sufficient conditions for existence of J-spectral factorization for para-Hermitian rational matrix functions.* Automatica, 39:1935–1939, 2003.

[89] Rice J. R., *A theory of condition.* SIAM J. Numer. Anal., 3(2):287–310, 1966.

[90] Rosenbrock H., *State-Space and Multivariable Theory.* Wiley, New York, 1970.

[91] Scilab, *Launch of the Scilab Consortium dedicated to scientific computing.* INRIA and Ecole Nationale des Ponts et Chaussées, France, May 2003.

[92] Šebek, M., *An algorithm for spectral factorization of polynomial matrices with any signature*. Memorandum 912, Universiteit Twente, The Netherlands, 1990.

[93] Skeel R., *Roundoff error and the Patriot missile*. SIAM News, 25(4):11, 1992.

[94] Stefanidis P., Papliński A. P. and Gibbard M. J., *Numerical operations with polynomial matrices: Application to multi-variable dynamic compensator design*. Lecture Notes in Control and Information Sciences, 171, Springer Verlag, New York, 1992.

[95] Stefanovski J., *Polynomial J-spectral factorization in minimal state-space*. Automatica, 39:1893–1901, 2003.

[96] Stefanovski J., *Discrete J-spectral factorization of possibly singular polynomial matrices*. System Control Lett., 53:127–140, 2004.

[97] Stetter H. J., *Numerical polynomial algebra: concepts and algorithms*. Asian Technology Conference in Matemathics, Chiang Mai, Thailand, 2000.

[98] Stetter H. J., *Numerical Polynomial Algebra*. SIAM, Philadelphia, 2004.

[99] Stewart G. W., *Matrix Algorithms*. SIAM, Philadelphia, 1998.

[100] Tisseur F., *Backward error and condition of polynomial eigenvalue problems*. Linear Algebra Appl. 309:339–361, 2000.

[101] Tisseur F. and Meerbergen K., *The quadratic eigenvalue problem*. SIAM Review, 43:235–286, 2001.

[102] Trentelman H. L. and Rapisarda P., *New algorithms for polynomial J-spectral factorization*. Math. Control Signal Systems, 12:24–61, 1999.

[103] Tuel W. G., *Computer algorithm for spectral factorization of rational matrices*. IBM Journal, 163–170, 1968.

[104] Turing A. M., *Rounding-off errors in matrix processes*. Quart. J. Mech. Appl. Math., 1:287–308, 1948.

[105] Van Dooren P. M., *The computation of Kronecker's canonical form of a singular pencil*. Linear Algebra Appl., 27,103–140, 1979.

[106] Van Dooren P. M., Dewilde P. and Vandewalle J., *On the determination of the Smith-MacMillan form of a rational matrix from its Laurent expansion*. IEEE Trans. Circuit Systems, 26:180–189, 1979.

[107] Van Dooren P. M. and Dewilde P., *The eigenstructure of an arbitrary polynomial matrix: computational aspects*. Linear Algebra Appl., 50:545–580, 1983.

[108] Van Dooren P. M., *The basics of developing numerical algorithms*. IEEE Control Systems Magazine, 24:18–27, 2004.

[109] Vardulakis A. I. G., *Linear Multivariable Control. Algebraic Analysis and Synthesis Methods*. Wiley, Chichester, 1991.

[110] Varga A., *A Descriptor System Toolbox for Maltab*, IEEE International Symposium on Computer Aided Control System Design, Anchorage, Alaska, 2000.

[111] Varga A., *Computation of least order solutions of linear rational equations*. International Symposium on Mathematical Theory of Networks and Systems, Leuven, Belgium, 2004.

[112] Von Neumann J. and Goldstine H. H., *Numerical inverting of matrices of high order*. Bull. Amer. Math. Soc., 53:1021–1099, 1947.

[113] Waterloo Maple Inc., *Maple*, Version 9.5, Waterloo, Canada, 2005.

[114] Wiener N., *Extrapolation, Interpolation and Smoothing of Stationary Time Series*. Wiley, New York, 1949.

[115] Wilkinson J. H., *The Algebraic Eigenvalue Problem*. Oxford University Press, 1965.

[116] Wilkinson J. H., *Rounding Errors in Algebraic Processes*. Dover, New York, 1994.

[117] Wolfram Research Inc., *Mathematica*, Version 5.1, Champaign, Illinois, 2005.

[118] W. M. Wonham and A. S. Morse, *Decoupling and pole assignment in linear multivariable systems: A geometric approach*, SIAM J. Control Optim., 8, pp. 1–18, 1970.

[119] Zeng Z., *Computing multiple roots of inexact polynomials*. Math. Comput., 64:869–903, 2005.

[120] Zha H., *A componentwise perturbation analysis of the QR decomposition*. SIAM J. Matrix Anal. Appl., 14(4):1124–1131, 1993.

[121] Zhang H., *Numerical condition of polynomials in different forms*. Electron. Trans. Numer. Anal., 12:66–87, 2001.

[122] Zúñiga J. C., Ruiz-Léon J. and Henrion D., *Algorithm for decoupling and complete pole assignement of linear multivariable systems*. European Control Conference, Cambridge, UK, 2003.

[123] Zúñiga J. C. and Henrion D., *Comparison of algorithms for computing infinite structural indices of polynomial matrices*. European Control Conference, Cambridge, UK, 2003.

[124] Zúñiga J. C. and Henrion D., *Block Toeplitz methods in polynomial matrix computations*. International Symposium on Mathematical Theory of Networks and Systems, Leuven, Belgium, 2004.

[125] Zúñiga J. C. and Henrion D., *A Toeplitz algorithm for the polynomial J-spectral factorization*. IFAC Symposium on System, Structure and Control, Oaxaca, Mexico, 2004. Journal version submitted to Automatica.

[126] Zúñiga J. C. and Henrion D., *On the application of displacement structure methods to obtain null-spaces of polynomial matrices*. IEEE Conference on Decision and Control, Paradise Island, Bahamas, 2004.

[127] Zúñiga J. C. and Henrion D., *Numerical stability of block Toeplitz algorithms in polynomial matrix computations*. IFAC World Congress on Automatic Control, Prague, Czech Republic, 2005.

[128] Zúñiga J. C. and Henrion D., *Block Toeplitz algorithms for polynomial matrix null-space computation*. Submitted to SIAM J. Matrix Anal. Appl., December 2004.

Index